Lista 3

César A. Galvão - 190011572

Gabriela Carneiro - 180120816

João Vitor Vasconcelos - 170126064

Índice

Questão 6

Estudar o texto de Kneusel (2022) sobre o algoritmo Gradiente Descendente[1] e apresentar um resumo com exemplos em `R`.

Gradiente Descendente é uma técnica de otimização fundamental no campo do aprendizado de máquina, usada para minimizar uma função de custo ajustando iterativamente os parâmetros do modelo. Esse algoritmo calcula o gradiente da função de custo e atualiza os parâmetros na direção oposta ao gradiente, visando reduzir o erro.

Gradiente Descendente

O gradiente é o vetor de derivadas parciais da função de custo em relação a cada parâmetro. A atualização dos parâmetros segue a fórmula:

$$\theta_{\text{novo}} = \theta_{\text{antigo}} - \eta \times \nabla F(\theta_{\text{antigo}}) \tag{1}$$

onde η é a taxa de aprendizado, e ∇F representa o gradiente.

Para o código a seguir, considera-se

$$y_i = 2x_i + \epsilon_i \tag{2}$$

e o Erro Quadrático Médio (MSE) como função de custo:

$$F(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \theta x_i)^2 \tag{3}$$

de modo que

$$\nabla F(\theta) = -\frac{2}{n}\sum_{i=1}^{n}(y_i - \theta x_i)x_i\frac{d}{d\theta}. \tag{4}$$

O passo no sentido do gradiente descendente escolhido (*learning rate*) foi de 0.0001 e o algoritmo foi executado por 1000 iterações (*epochs*).

```r
# Definir semente para garantir a reprodutibilidade
set.seed(123)

# Simulação de dados
x <- 1:100
y <- 2 * x + rnorm(100, mean = 0, sd = 10)  # y é uma função linear de x com ruído
↪  adicionado
```

[1] Ronald T. Kneusel (2022) Math for Deep Learning.

```r
theta <- runif(1, min = -2, max = 2)  # Inicializar theta aleatoriamente
learning_rate <- 0.0001  # Definir a taxa de aprendizado

# Função para calcular o gradiente do erro quadrático médio
mse_gradient <- function(x, y, theta) {
  gradient <- -2/length(y) * sum((y - theta * x) * x)
  return(gradient)
}

# Loop para atualizar o parâmetro theta usando Gradiente Descendente
for (i in 1:1000) {
  grad <- mse_gradient(x, y, theta)
  theta <- theta - learning_rate * grad
}
```

```r
# Criar um data frame para a visualização com ggplot2
data <- data.frame(x, y)

# Visualização do ajuste usando ggplot2
ggplot(data, aes(x = x, y = y)) +
  geom_point() +  geom_abline(intercept = 0, slope = theta, color = "red", size = 1)
  ↪  +  labs(x = "Variável Independente (x)",
       y = "Variável Dependente (y)") +
  theme_minimal()
```

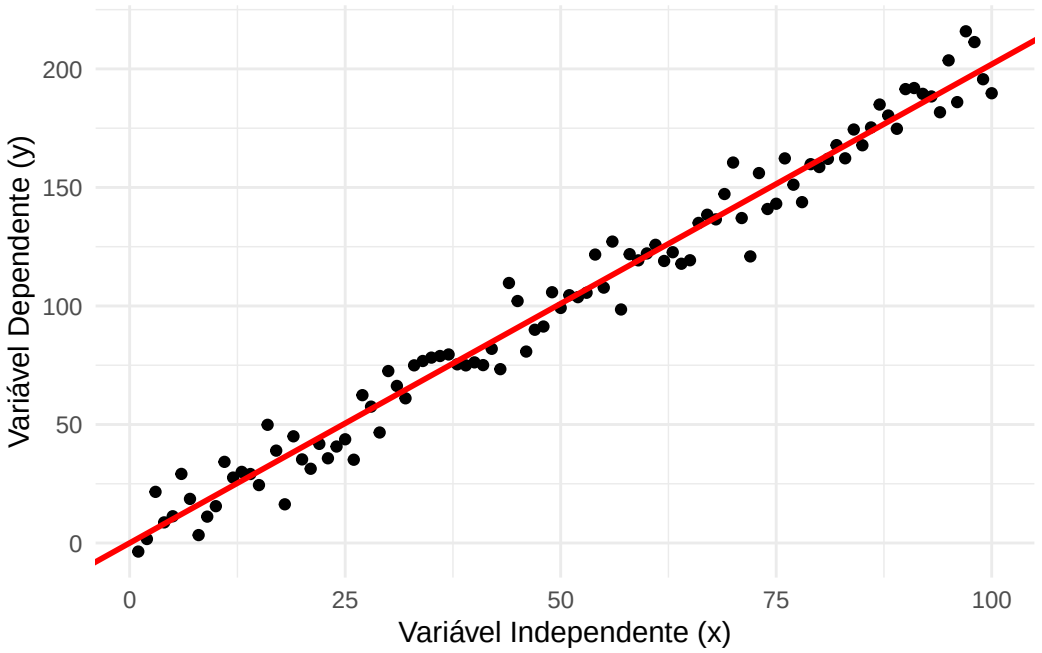


Figura 1: Gradiente Descendente para Regressão Linear

O parâmetro θ converge para 2.0197, um valor próximo de 2, que é a inclinação real da relação entre

x e y. A linha vermelha no gráfico representa a linha de regressão ajustada, que mostra como o modelo linear com a inclinação estimada se ajusta aos dados gerados. O resultado ilustra a eficácia do Gradiente Descendente em encontrar a inclinação que minimiza o erro quadrático médio entre as previsões e os valores reais.

Algoritmo Perceptron

O Algoritmo Perceptron é um classificador binário que ajusta os pesos com base nos erros de classificação.

Principais Conceitos

- **Classificação Binária**: O Perceptron é capaz de separar duas classes linearmente separáveis através de uma função de decisão linear.
- **Aprendizado Supervisionado**: Utiliza rótulos conhecidos para aprender a fronteira de decisão, ajustando iterativamente os pesos.

Formulação Matemática

O Perceptron ajusta os pesos $\boldsymbol{w}$ e o viés b usando a regra de atualização:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta(y_i - \hat{y}_i)\boldsymbol{x}_i$$

onde η é a taxa de aprendizado, y_i é o rótulo verdadeiro, $\hat{y}_i$ é a previsão, e $\boldsymbol{x}_i$ são as características de entrada.

Exemplo

```
# Carregar a biblioteca ggplot2
library(ggplot2)

# Define uma semente aleatória para garantir que os resultados sejam reprodutíveis
set.seed(42)

# Gera uma matriz 100x2 de números aleatórios normalmente distribuídos
x <- matrix(rnorm(200), 100, 2)

# Cria um vetor com 100 elementos, sendo os primeiros 50 elementos -1 e os últimos
↪  50 elementos 1
y <- c(rep(-1, 50), rep(1, 50))

# Aumenta todos os valores nas duas colunas de 'x' para as entradas onde 'y' é 1
x[y == 1, ] <- x[y == 1, ] + 1

# Converte os arrays 'x' e 'y' em um data frame chamado 'df' e transforma 'y' em um
↪  fator
df <- data.frame(x1 = x[, 1], x2 = x[, 2], class = as.factor(y))

# Inicializa um vetor de pesos com zeros, com um elemento a mais do que o número de
↪  colunas em 'x'
weights <- rep(0, ncol(x) + 1)
```

```r
# Define a função Perceptron para ajustar os pesos com base nas entradas 'x', 'y',
↪  pesos existentes e taxa de aprendizado
perceptron <- function(x, y, weights, learning_rate = 0.01) {
  for (i in 1:length(y)) {
    # Verifica se a classificação atual está correta, ajusta os pesos se estiver
    ↪  errada
    if (y[i] * (crossprod(weights, c(1, x[i, ]))) <= 0) {
      weights <- weights + learning_rate * y[i] * c(1, x[i, ])
    }
  }
  return(weights)
}

# Treina o modelo Perceptron por 10 épocas, atualizando os pesos a cada iteração
for (epoch in 1:10) {
  weights <- perceptron(x, y, weights)
}

# Calcula a inclinação e a interceptação da linha de decisão baseada nos pesos
↪  finais
slope <- -weights[2]/weights[3]
intercept <- -weights[1]/weights[3]
```

```r
# Configura o gráfico ggplot para 'df', mapeando 'x1' e 'x2' para os eixos x e y, e
↪  'class' para a cor
ggplot(df, aes(x = x1, y = x2, color = class)) +
  geom_point() +  # Adiciona pontos ao gráfico para representar as observações
  geom_abline(slope = slope, intercept = intercept, color = "green", linewidth = 1)
  ↪  +  # Adiciona uma linha de decisão
  labs(x = "Feature 1", y = "Feature 2") +  # Define os títulos e rótulos dos eixos
  scale_color_manual(values = c("red", "blue")) +  # Define cores manuais para as
  ↪  classes
  theme_minimal()  # Aplica um tema minimalista ao gráfico
```

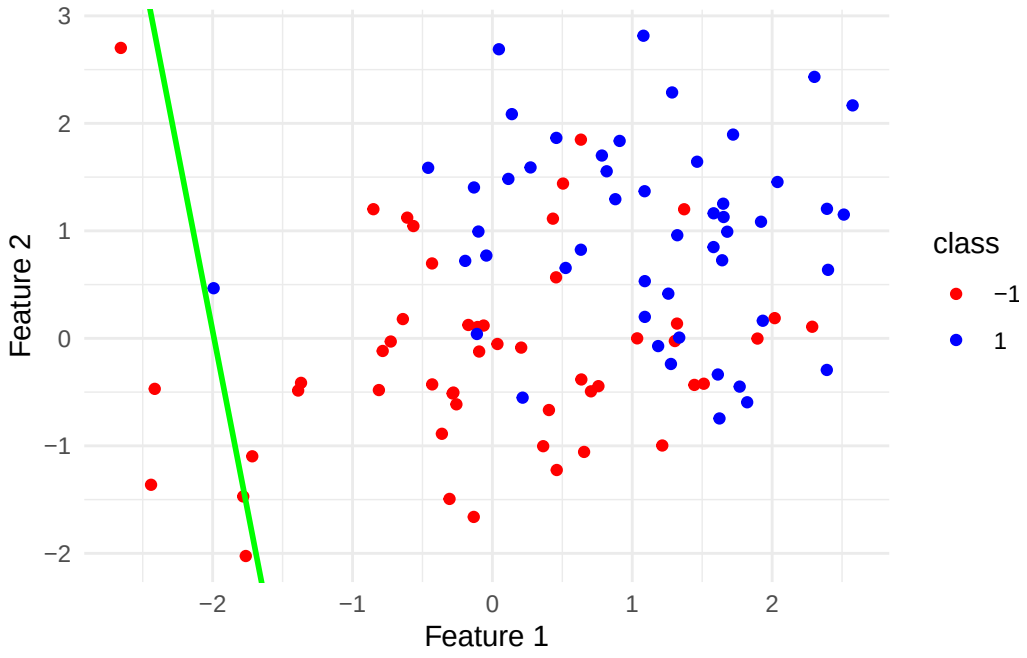


Figura 2: Classificação do Perceptron

O Perceptron ajusta uma fronteira de decisão (linha verde) que tenta separar os pontos de dados em duas classes (vermelho e azul). Os pesos são ajustados baseados em erros de classificação, e o resultado visual mostra como a linha foi capaz de separar as duas classes após o treinamento.

O resultado do algoritmo Perceptron inclui:

- **Modelo de Classificação Treinado**: O Perceptron é um classificador linear binário que aprende a distinguir entre duas classes (-1 e 1 nesse caso). O treinamento ajusta os pesos associados a cada atributo dos dados (e um peso de viés) de modo que o modelo possa corretamente classificar novos exemplos baseado em suas características.
- **Pesos Ajustados**: Ao final do treinamento, os pesos representam os parâmetros de um hiperplano que melhor separa as duas classes no espaço de características. Esses pesos determinam como as entradas (características dos dados) são linearmente combinadas para fazer uma previsão de classe.
- **Fronteira de Decisão**: No espaço bidimensional do exemplo, essa fronteira é representada por uma linha reta. A posição e orientação dessa linha são diretamente determinadas pelos pesos resultantes do treinamento.

Critério de Fisher

O Critério de Fisher, ou Análise Discriminante Linear (LDA), é uma técnica que busca maximizar a separação entre diferentes classes enquanto minimiza a dispersão dentro de cada classe. É usado primariamente para reduzir a dimensionalidade dos dados antes da classificação, melhorando assim a eficácia dos modelos de aprendizado de máquina em ambientes de alta dimensionalidade.

Principais Conceitos

- **Variância entre as Classes**: Maximiza a distância entre as médias das classes.

- **Variância dentro das Classes**: Minimiza a dispersão dos dados dentro de cada classe.

Formulação Matemática

O vetor de projeção $\mathbf{w}$ é determinado maximizando a razão:

$$J(\mathbf{w}) = \frac{\sigma^2_{\text{entre}}}{\sigma^2_{\text{dentro}}}$$

onde σ^2_{entre} e σ^2_{dentro} representam, respectivamente, a variância entre as médias das classes e a soma das variâncias dentro de cada classe projetada.

Exemplo

```r
# Filtrar o conjunto de dados iris para excluir a espécie 'virginica'
iris_subset <- droplevels(iris[iris$Species != "virginica", ])

# Ajustar o modelo LDA ao subconjunto de dados
model <- lda(Species ~ ., data = iris_subset)

# Obter as projeções LDA usando predict
projection <- predict(model)$x  # Isto obtém os valores discriminantes diretamente

# Converter projeções e espécies em um dataframe para uso no ggplot
plot_data <- data.frame(Index = 1:nrow(projection), Projection = projection[,1],
↪  Species = iris_subset$Species)
```

```r
# Criar o gráfico usando ggplot
ggplot(plot_data, aes(x = Projection, y = Index, color = Species)) +
  geom_point(alpha = 0.6) +  # Use alpha para ajustar a transparência, se necessário
  labs(x = "Projection", y = "Index") +
  scale_color_brewer(palette = "Set1", name = "Species") +
  theme_minimal()
```

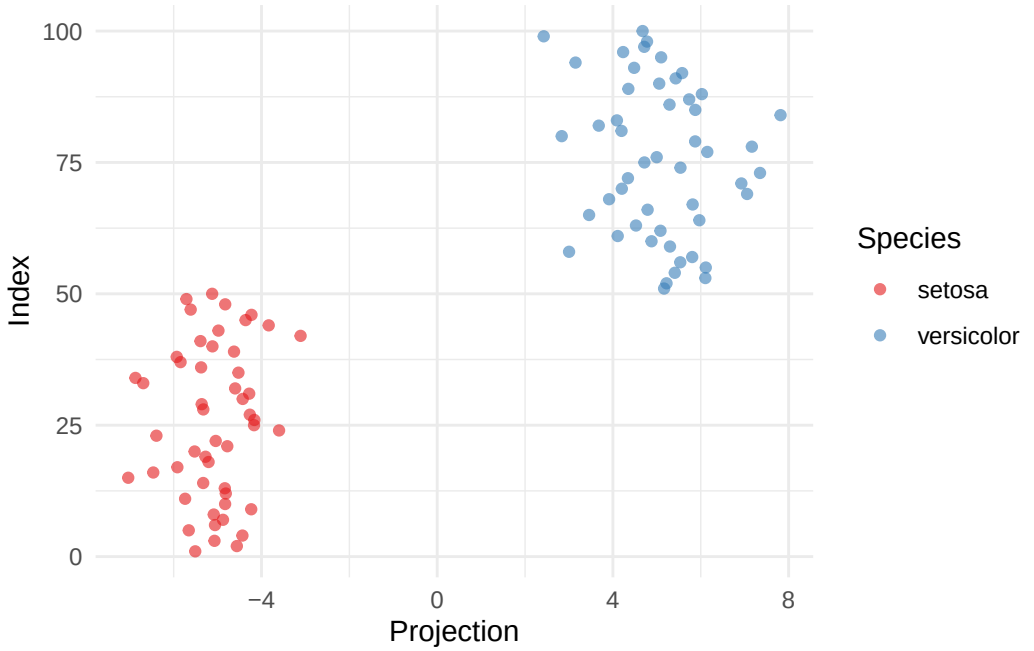


Figura 3: Projeção LDA com Critério de Fisher

O gráfico mostra a eficácia da projeção LDA em separar duas espécies de íris, utilizando o Critério de Fisher para maximizar a distância entre as classes.

Critério de Mínimos Quadrados

O Critério de Mínimos Quadrados é usado para estimar os coeficientes de um modelo de regressão. O objetivo é minimizar a soma dos quadrados das diferenças entre os valores observados e os valores estimados pelo modelo, garantindo assim a melhor adequação possível dos dados ao modelo encontrado.

Principais Conceitos

- **Erro Quadrático**: Mede a diferença ao quadrado entre os valores observados e os preditos pelo modelo.
- **Ajuste de Modelo**: O critério busca parâmetros que reduzam ao mínimo o erro quadrático total, refletindo a melhor previsão possível dada a variabilidade dos dados.

Formulação Matemática

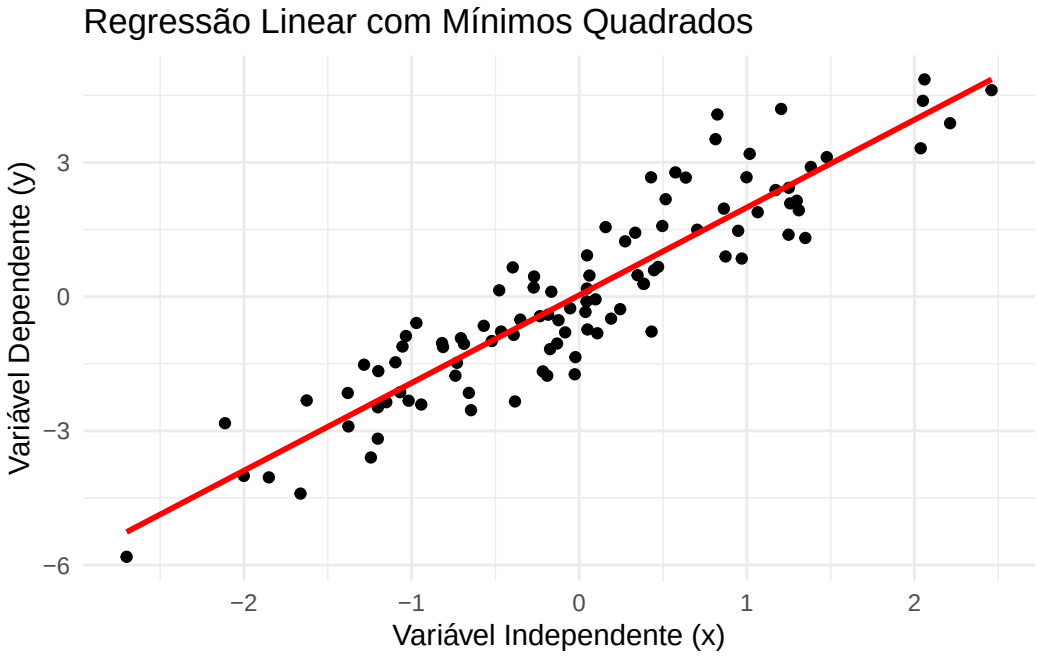
A estimativa dos coeficientes do modelo de regressão é realizada através da solução da equação:

$$\beta = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$$

onde $\mathbf{X}$ representa a matriz de dados (com uma coluna adicional de uns para o intercepto), β são os coeficientes a serem estimados, e $\mathbf{y}$ é o vetor de variáveis dependentes.

Exemplo

```r
# Gerar 100 números aleatórios normalmente distribuídos como variável independente
↪  'x'
x <- rnorm(100)

# Criar a variável dependente 'y' como uma função linear de 'x' mais ruído gaussiano
y <- 2 * x + rnorm(100)

# Criar um data frame a partir de x e y
data <- data.frame(x, y)

# Ajustar um modelo de regressão linear onde 'y' é predito por 'x'
model <- lm(y ~ x, data = data)

# Criar um gráfico de dispersão usando ggplot
MQ <- ggplot(data, aes(x = x, y = y)) +
  geom_point() +  # Adicionar pontos ao gráfico
  geom_smooth(method = "lm", se = FALSE, color = "red", size = 1) +  # Adicionar
  ↪  linha de regressão
  labs(title = "Regressão Linear com Mínimos Quadrados", x = "Variável Independente
  ↪  (x)", y = "Variável Dependente (y)") +
  theme_minimal()  # Utilizar um tema minimalista para o gráfico

# Exibir o gráfico
print(MQ)
```



A linha vermelha mostra a melhor ajuste linear, minimizando as diferenças quadradas entre os valores observados e preditos, demonstrando a precisão do modelo na estimativa da relação entre as variáveis.

Questão 7

Estudar e apresentar um exemplo utilizando as notas sobre Perceptron no R descritas em https://rpubs.com/FaiHas/197581.

O autor apresenta o algoritmo de Frank Rosenblatt para classificação. O exemplo apresentado considera uma base de dados contendo uma lista de pesos e rótulos indicando a espécie de íris do banco de dados `datasets::iris`. O perceptron recebe a base de dados e "aprende" a identificar as classes corretamente com base nas características da observação.

A base de dados `iris` original é composta por 150 observações de 4 variáveis (comprimento e largura da sépala e pétala) e 1 variável de classe (espécie da íris). O autor utiliza um subconjunto de 100 observações aleatórias da base de dados e cria um vetor de rótulos binários para as espécies "setosa" e "versicolor".

```
irissubdf <- iris[1:100, c(1, 3, 5)]
names(irissubdf) <- c("sepal", "petal", "species")
head(irissubdf) %>%
  knitr::kable()
```

Tabela 1: Subconjunto de Dados Iris

sepal	petal	species
5.1	1.4	setosa
4.9	1.4	setosa
4.7	1.3	setosa
4.6	1.5	setosa
5.0	1.4	setosa
5.4	1.7	setosa

O subconjunto de dados é exibido na Figura 4 a seguir.

```
ggplot(irissubdf, aes(x = sepal, y = petal,
                      colour=species, shape= species)) +
       geom_point(size = 3) +
       labs(x = "Comp. Sépala", y = "Comp. Pétala")+
  theme_bw()+
  theme(legend.title = element_blank())
```

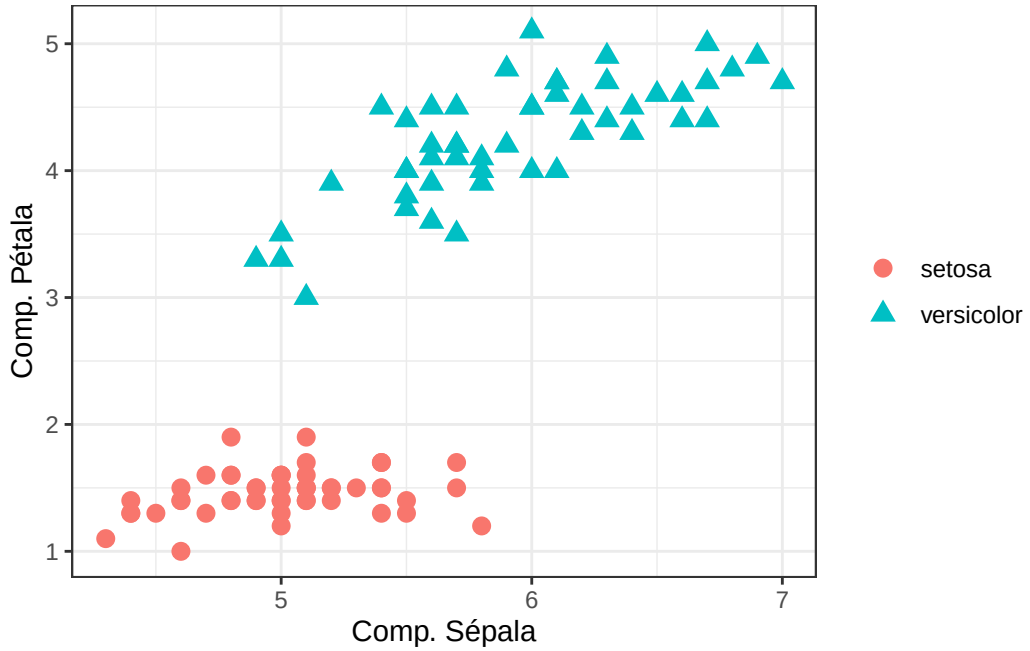


Figura 4: Espécies em função de comprimento de sépala e pétala

Em seguida, o autor adiciona uma quarta variável ao subconjunto de dados representando o rótulo. Se a espécie for "setosa", o rótulo é 1; caso contrário, o rótulo é -1. Em seguida, as características são atribuídas ao objeto `x` e os rótulos ao objeto `y`.

```
irissubdf <- irissubdf %>%
  mutate(label = case_when(
    species == "setosa" ~ 1,
    species == "versicolor" ~ -1
  ))

x <- irissubdf %>% dplyr::select(sepal, petal)
y <- irissubdf %>% dplyr::select(label) %>% pull()
```

Finalmente, o algoritmo perceptron é implementado na mesma forma que foi apresentado nas notas de aula. A função que o executa recebe quatro argumentos: dados, rótulos, taxa de aprendizagem e número de iterações.

A função é inicializada com um vetor de pesos $\boldsymbol{w} = (w_1 = 0, w_2 = 0, b = 0)$ e um vetor nulo de erros do mesmo tamanho que o número de iterações. O algoritmo percorre o conjunto de dados e ajusta os pesos de acordo com a regra de atualização do perceptron em dois laços:

- o primeiro laço percorre o número de iterações;
- o segundo laço percorre o número de observações em cada iteração.

No algoritmo, $z = \boldsymbol{w}^\top \boldsymbol{x} + b$ é a previsão da observação com os pesos correntes, porém é feito elemento a elemento. Se $z < 0$, a previsão é -1; caso contrário, a previsão é 1. Os pesos são atualizados da seguinte forma para cada observação i:

$$\boldsymbol{w}^{(t)} \leftarrow \boldsymbol{w}^{(t-1)} + \eta(y_i - \hat{y}_i) \begin{bmatrix} 1 \\ \boldsymbol{x}_i \end{bmatrix}$$

em que η é a taxa de aprendizagem. Nota-se que o peso *não* é atualizado se a classificação for correta. O vetor

Depois de percorrer a base de dados completa em cada iteração, o erro é atualizado se a classificação for incorreta. O vetor final de erros da função indica quantas classificações erradas foram feitas em cada iteração.

Finalmente, uma pequena alteração foi feita na função para que seja retornada uma lista com dois elementos: o vetor de pesos e o vetor de erros.

```
perceptron <- function(x, y, eta, niter) {

        # initialize weight vector
        weight <- rep(0, dim(x)[2] + 1)
        errors <- rep(0, niter)


        # loop over number of epochs niter
        for (jj in 1:niter) {

                # loop through training data set
                for (ii in 1:length(y)) {

                        # Predict binary label using Heaviside activation
                        # function
                        z <- sum(weight[2:length(weight)] *
                                        as.numeric(x[ii, ])) + weight[1]
                        if(z < 0) {
                                ypred <- -1
                        } else {
                                ypred <- 1
                        }

                        # Change weight - the formula doesn't do anything
                        # if the predicted value is correct
                        weightdiff <- eta * (y[ii] - ypred) *
                                c(1, as.numeric(x[ii, ]))
                        weight <- weight + weightdiff

                        # Update error function
                        if ((y[ii] - ypred) != 0.0) {
                                errors[jj] <- errors[jj] + 1
                        }
```

```
                }
        }

        # weight to decide between the two species
        return(list(
          w = weight,
          erros = errors)
          )
}
```

Com o subconjunto escolhido pelo autor, obtemos o vetor de pesos

```
pesos_iris<- perceptron(x, y, 1, 10)
```

$$\boldsymbol{w} = \begin{bmatrix} 4 \\ 7 \end{bmatrix} \quad b = -18.4 \tag{5}$$

e a quantidade de erros por iterações, apresentada na Figura 5 a seguir, um pouco diferente do que o autor exibe.

```
ggplot(data = as_tibble(pesos_iris$erros), aes(x = 1:length(pesos_iris$erros), y =
↪  pesos_iris$erros)) +
  geom_line(color = "red") +
  labs(x = "Iterações", y = "Erros") +
  theme_bw() +
  theme(
    panel.grid.major.x = element_blank(), # Remove major x grids
    panel.grid.minor.x = element_blank(), # Remove minor x grids
    panel.grid.minor.y = element_blank()  # Remove minor y grids
  )
```

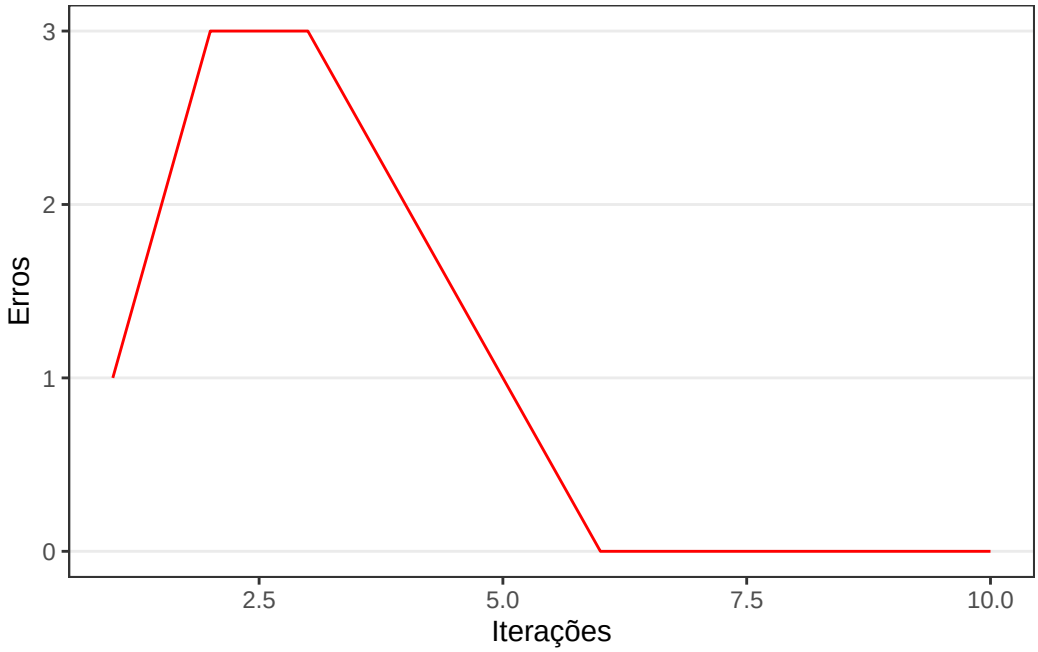


Figura 5: Erros por Iterações do algoritmo Perceptron

O autor finalmente tece comentários sobre classificação multiclasse usando o perceptron. Trata-se de um algoritmo com baixa capacidade de representação, sendo adequado apenas para problemas de classificação binária, portanto seria necessário tanto mais características das observações quanto procedimentos de classificação do tipo Classe 1 *versus* Demais.

Algo que se é necessário destacar também é que, enquanto o erro de classificação convergiu para zero no exemplo apresentado, isso não é garantido para todos os problemas. Com as três espécies, conforme a Figura 6 a seguir, duas não são linearmente separáveis entre si, de modo que o erro converge para um valor superior a zero para esta configuração do algoritmo.

```
irisdata <- iris[, c(1, 3, 5)]
names(irisdata) <- c("sepal", "petal", "species")

ggplot(irisdata, aes(x = sepal, y = petal,
                     colour=species, shape= species)) +
        geom_point(size = 3) +
        labs(x = "Comp. Sépala", y = "Comp. Pétala")+
  theme_bw()+
  theme(legend.title = element_blank())
```

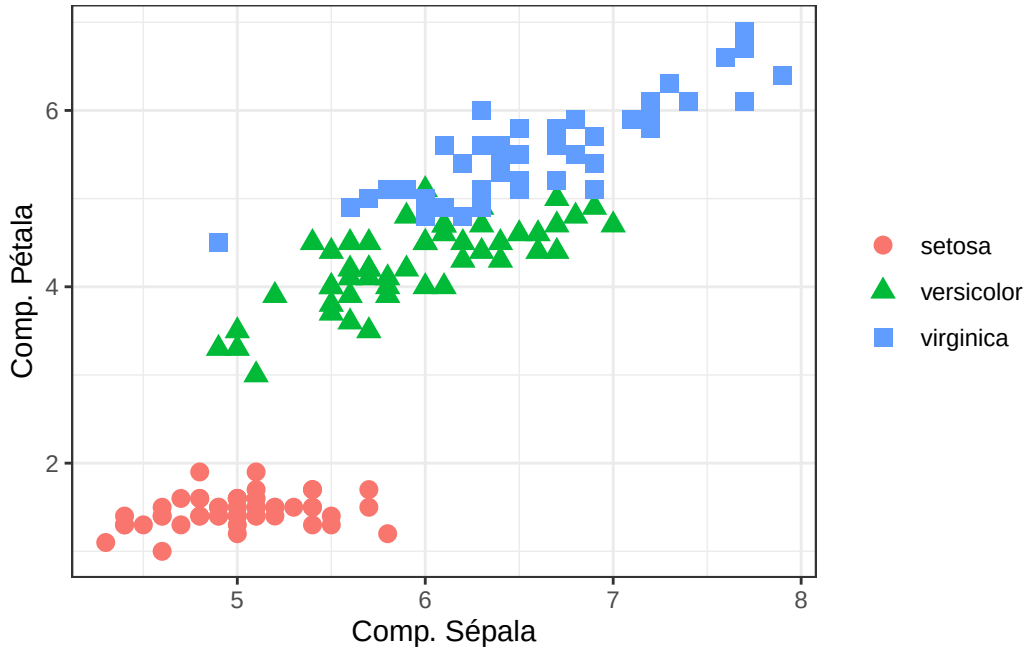


Figura 6: Espécies em função de comprimento de sépala e pétala usando a base iris completa

Questão 8

a) Estudar e apresentar a função perceptron do pacote `mlpack` do R.
b) Verificar se existe material adicional (instruções em sites, pacotes, artigos) sobre classificação utilizando o algoritmo perceptron no R e apresentar exemplos.

---

item a)

De acordo com a documentação da função `mlpack::perceptron()`[2],

"Este programa implementa um perceptron, que é uma rede neural de nível único. O perceptron faz suas previsões com base em uma função preditora linear que combina um conjunto de pesos com o vetor de características. A regra de aprendizado do perceptron é capaz de convergir, dadas iterações suficientes (especificadas usando o parâmetro max_iterations), se os dados fornecidos forem linearmente separáveis. O perceptron é parametrizado por uma matriz de vetores de peso que denotam os pesos numéricos da rede neural.

Este programa permite carregar um perceptron a partir de um modelo (via o parâmetro input_model) ou treinar um perceptron com dados de treinamento (via o parâmetro training), ou ambas as coisas ao mesmo tempo. Além disso, este programa permite a classificação em um conjunto de dados de teste (via o parâmetro test) e os resultados da classificação no conjunto de teste podem ser salvos com o parâmetro de saída predictions. O modelo de perceptron pode ser salvo com o parâmetro de saída output_model."

São 7 os possíveis argumentos de entrada:

- **check_input_matrices**
  - *tipo*: lógico
  - *descrição*: Se especificado, a matriz de entrada é verificada para valores NaN e inf; uma exceção é lançada se algum for encontrado.
  - *padrão*: FALSE
- **input_model**
  - *tipo*: PerceptronModel
  - *descrição*: Modelo de perceptron de entrada.
  - *padrão*: NA
- **labels**
  - *tipo*: vetor de inteiros
  - *descrição*: Uma matriz contendo os rótulos para o conjunto de treinamento.
  - *padrão*: matrix(integer(), 0, 0)
- **max_iterations**
  - *tipo*: inteiro
  - *descrição*: O número máximo de iterações que o perceptron deve executar.
  - *padrão*: 1000
- **test**
  - *tipo*: matriz numérica
  - *descrição*: Uma matriz contendo o conjunto de teste.
  - *padrão*: matrix(numeric(), 0, 0)
- **training**
  - *tipo*: matriz numérica
  - *descrição*: Uma matriz contendo o conjunto de treinamento.

[2] https://www.mlpack.org/doc/mlpack-git/r_documentation.html#perceptron

  - *padrão*: matrix(numeric(), 0, 0)
- **verbose**
  - *tipo*: lógico
  - *descrição*: Exibir mensagens informativas e a lista completa de parâmetros e temporizadores no final da execução.
  - *padrão*: FALSE

Os resultados são apresentados em uma lista com os seguintes elementos:

- **output**
  - *tipo*: vetor de inteiros
  - *descrição*: A matriz na qual os rótulos previstos para o conjunto de teste serão escritos.
- **output_model**
  - *tipo*: PerceptronModel
  - *descrição*: Saída para o modelo de perceptron treinado.
- **predictions**
  - *tipo*: vetor de inteiros
  - *descrição*: A matriz na qual os rótulos previstos para o conjunto de teste serão escritos.

item b)

Enquanto o próprio mlpack.org já traz a documentaçào da função, que é a mesma do CRAN, os próprios autores têm artigos publicados sobre a elaboração do pacote e seu desempenho (Edel, Soni, e Curtin 2014; Curtin e Edel 2017).

No entanto, outros materiais, inclusive videos do YouTube, não foram identificados. Em vez disso, os resultados apresentam tutoriais e teoria acerca do *multilayer perceptron* — que é a própria Deep Learning.

Questão 9 - Critério de Fisher

A Análise Discriminante Linear (*Linear Discriminant Analysis* – LDA) pode ser considerada uma extensão do critério de Fisher para reconhecimento de padrões, sendo esse um método estatístico que busca encontrar uma combinação linear dos recursos que melhor discrimina entre duas ou mais classes.

A utilização dessa técnica no `R` tem suas vantagens pela facilidade com a qual pode ser implementada e a versatilidade ao se integrar com outras funções estatísticas do R, mas pode acabar sofrendo com conjuntos de dados muito grandes ou necessidades computacionais muito altas.

Exemplo

A seguir temos um exemplo da implementação de LDA no banco de dados Iris, composto classificações de 3 espécies de flores e medidas em centímetros das variáveis comprimento e largura da sépala e comprimento e largura da pétala.

```
data(iris)
dados <- iris

modelo_discriminante <- lda( Species ~ Sepal.Length +Sepal.Width+ Petal.Length +
↪  Petal.Width, data = dados )

data.predic <-predict(modelo_discriminante,newdata = dados[,1:4])$class
```

A Tabela 2 a seguir apresenta a quantidade de classificações corretas e a quantidade de classificações erradas a partir do nosso modelo ajustado.

```
table(data.predic, dados[,5]) %>%
  knitr::kable()
```

Tabela 2: Tabela de Classificações Corretas e Incorretas do LDA

	setosa	versicolor	virginica
setosa	50	0	0
versicolor	0	48	1
virginica	0	2	49

A Figura 7 a seguir nos mostra como, a partir dos Discriminates lineares estimados do nosso modelo LDA, as divisões de classes foram feitas. O eixo LDA1 representa a direção da máxima separação entre as classes, enquanto o eixo LDA2 representa a direção da segunda maior separação.

O diagrama de dispersão demonstra uma clara separação entre as três classes de flores de íris. As flores setosa estão agrupadas no canto superior direito do gráfico, as flores versicolor e flores virginica estão agrupadas no canto inferior esquerdo distribuídas entre os dois aglomerados. As flores setosa são as mais bem agrupadas, indicando menor variabilidade dentro dessa classe. As flores versicolor e virgínica estão mais espalhadas, sugerindo maior variabilidade dentro dessas duas classes

O ponto marcado por X em cada classe de plantas indica a sua média geral. Assim,

- *Setosa*: indica que a sua média tem um valor LDA1 positivo e um valor LDA2 negativo tendo a maior diferença em relação à média geral entre as três classes.

- *Virginica*: indica que a sua média tem um valor LDA1 negativo e um valor LDA2 positivo tendo a segunda maior diferença em relação à média geral entre as três classes.
- *Versicolor*: indica que a sua média tem um valor negativo tanto para LD1 quanto para LD2, mas perto de 0 tendo a menor diferença em relaçao a média geral.

```r
# Obter as predições do modelo LDA
lda_pred <- predict(modelo_discriminante)

# Adicionar as projeções ao dataframe original
iris$LD1 <- lda_pred$x[,1]
iris$LD2 <- lda_pred$x[,2]

# Definir cores para as espécies
species_colors <- c("setosa" = "red", "versicolor" = "blue", "virginica" = "green")

# Plotar as amostras projetadas nas componentes discriminantes
plot(iris$LD1, iris$LD2, col = species_colors[iris$Species], pch = 16, xlab = "LD1",
↪  ylab = "LD2")

# Adicionar a legenda ao gráfico
legend("topright", legend = levels(iris$Species), col = c("red", "blue", "green"),
↪  pch = 16)

# Calcular as médias das projeções das classes
class_means <- aggregate(cbind(LD1, LD2) ~ Species, data = iris, FUN = mean)

# Adicionar pontos representando as médias das classes
points(class_means$LD1, class_means$LD2, pch = 4, cex = 2, lwd = 2, col = "black")

# Adicionar linhas de separação (limiares) no gráfico
abline(v = mean(class_means$LD1), col = "black", lty = 2)  # LD1
abline(h = mean(class_means$LD2), col = "black", lty = 2)  # LD2
```

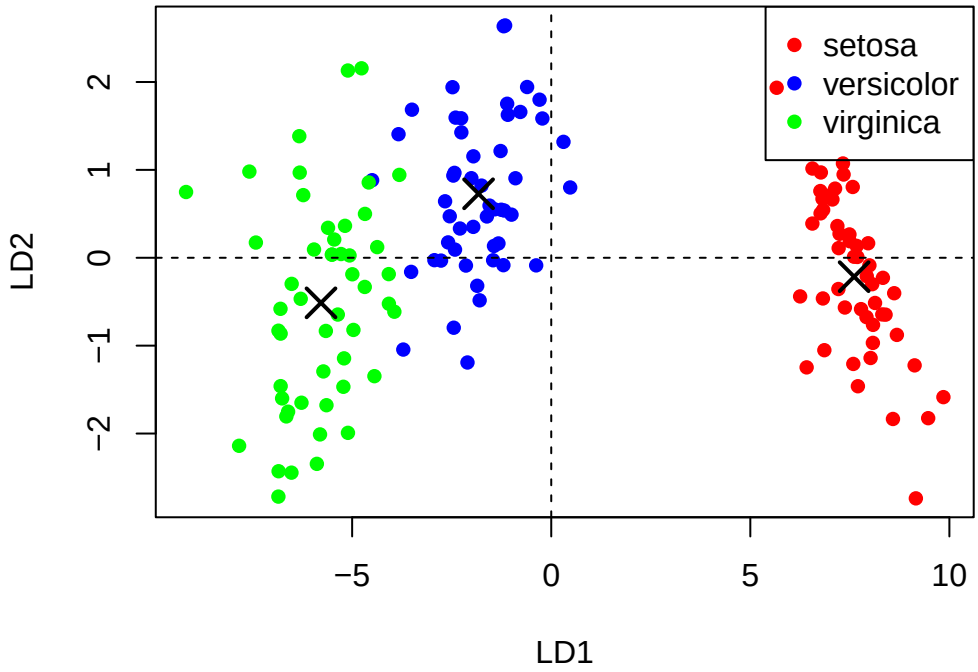


Figura 7: Gráfico de Classificação do LDA para o Conjunto de Dados Iris

Questão 10 - Critério de Minimos Quadrados

No R, a utilização do critério de Mínimos Quadrados pode ser exemplificada por meio de regressão linear e modelos lineares generalizados, utilizando as funções lm() e glm(), respectivamente. A implementação deste critério no R oferece várias vantagens, incluindo uma ampla gama de funções que o utilizam como base, facilidade de uso devido à sintaxe intuitiva e uma vasta comunidade que compartilha informações e recursos sobre essas implementações. Contudo, assim como o critério de Fisher, o critério de Mínimos Quadrados enfrenta desafios, como dificuldade em lidar com conjuntos de dados muito grandes, questões de desempenho e limitações de memória computacional .

Exemplo

Vamos aplicar uma regressão logística usando o pacote stats. Primeiramente, ajustaremos uma regressão linear simples com x_1 e x_2 sendo variáveis simuladas de uma distribuição normal com médias 2 e 3, respectivamente, e y sendo uma combinação de x_1 e x_2 com uma distribuição normal de média 0, posteriormente transformada em uma variável binária. Embora este problema seja mais adequado para uma regressão logística, usaremos este exemplo para ilustrar o critério de Mínimos Quadrados.

```r
# Gerar dados de exemplo
set.seed(123)
n <- 100
x1 <- rnorm(n, mean = 2, sd = 1)
x2 <- rnorm(n, mean = 3, sd = 1)
y <- ifelse(x1 + x2 + rnorm(n) > 5, 1, 0)
dados <- data.frame(x1, x2, y)

# Ajustar o modelo de regressão linear
modelo_linear <- lm(y ~ x1 + x2, data = dados)
summary(modelo_linear)
```

```
Call:
lm(formula = y ~ x1 + x2, data = dados)

Residuals:
     Min       1Q   Median       3Q      Max
-0.66422 -0.32225 -0.02538  0.25805  0.84446

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.73012    0.15228  -4.794 5.89e-06 ***
x1           0.19884    0.04189   4.746 7.14e-06 ***
x2           0.29541    0.03955   7.470 3.53e-11 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.38 on 97 degrees of freedom
Multiple R-squared:  0.436, Adjusted R-squared:  0.4244
F-statistic:  37.5 on 2 and 97 DF,  p-value: 8.616e-13
```

```r
# Fazer previsões
probabilidades <- predict(modelo_linear,newdata = dados, type = "response")
```

```r
# Transformar as probabilidades em classes binárias
predicoes <- ifelse(probabilidades > 0.7, 1, 0)

# Avaliar a precisão
precisao <- mean(predicoes == dados$y)
```

Assim calculamos a precisão comparando as previsões com as classes verdadeiras, obtendo um grau de precisão de 0.79.

Referências

Curtin, Ryan R, e Marcus Edel. 2017. «Designing and building the mlpack open-source machine learning library». *arXiv preprint arXiv:1708.05279*.

Edel, Marcus, Anand Soni, e Ryan R Curtin. 2014. «An automatic benchmarking system». Em *NIPS 2014 Workshop on Software Engineering for Machine Learning*.