# Modelo de Previsao de Ocorrencia de Diabetes - Versão 1.0

Giovanni Miranda

06/08/2023

## DEFINIÇÃO DO PROBLEMA

**Identificar pacientes com alta probabilidade de serem diagnosticados com diabetes, tendo, no mínimo, 75% de acurácia.**

## 0.0 - INSTALAÇÃO DOS PACOTES NECESSÁRIOS

**Instala pacote para tratamento de dados dplyr**

install.packages("dplyr")

**Instala pacote caTools para fazer a separação de dados de treino e teste do modelo**

install.packages("caTools")

**Instala pacote caret, para fazer o treino do modelo de classificação KNN**

install.packages("caret")

install.packages("e1071")

**Instala o pacote Naive bayes para fazer o treino do modelo com o algoritmo Naive Bayes**

install.packages("naivebayes")

**Instala o pacote Random Forest para fazer o treino do modelo com o algoritmo Random Forest**

install.packages("randomForest")

**Instala o pacote kernlab, para fazer o treino do modelo com SVM utilizando o kernel sigma**

install.packages("kernlab")

library(dplyr) library(caTools) library(caret) library(e1071) library(naivebayes) library(randomForest) library(kernlab)

# 1.0 OBTENÇÃO DOS DADOS

**Será utilizado o dataset público diabetes.csv, que contém dados de pacientes que desenvolveram diabetes e pacientes que não desenvolveram a doença.**

diabetes <- read.csv( file = "C:/Users/giova/Principal/DataScience/Escola_virtual_gov/analise_dados_R/scripts/modulo4_
)

head(diabetes)

# 2.0 PREPARAÇÃO DOS DADOS

**Verificando o tipo dos dados das colunas do dataset**

?str str(diabetes)

**Verificando se existem valores não preenchidos**

?colSums() colSums(is.na(diabetes))

**Verificando a proporção dos valores de cada categoria**

?table table(diabetes$Outcome)

**Alterando o tipo da coluna "Outcome" que é int para factor**

diabetes$Outcome <- as.factor(diabetes$Outcome)

**Verificando valores min, max, média, mediana...**

summary(diabetes$Insulin)

**Criando o gráfico de boxplot para cada coluna do dataset**

boxplot(diabetes)

**Criando o boxplot apenas da coluna "Insulin"**

boxplot(diabetes$Insulin)

**Criando um histograma da coluna "Insulin"**

hist(diabetes$Insulin)

**Obs: verificamos que a coluna "Insulin" possui muitos outliers. Isso pode prejudicar a acurácia do modelo. Precisaremes remover esses outliers. Utilizaremos o pacote dplyr para realizar algumas operações nesse sentido.**

**Filtrando o dataset por Insulin - Remoção de outliers**

diabetes2 <- diabetes %>% filter(Insulin <= 250)

boxplot(diabetes2$Insulin)

**Pronto: outliers da coluna Insulin foram devidamente removidos.**

# 3.0 ANÁLISE EXPLORATÓRIA

**Criação do boxplot para identificar outliers nas colunas do dataset**

boxplot(diabetes2)

### Análise: não temos outliers são absurdos nas demais colunas, como havia na coluna Insulin

**Criação de histogramas para visualizar a distribuição dos dados**

hist(diabetes2$Pregnancies$)hist($diabetes$2Age) hist(diabetes2$BMI)

**Visualizando os valores de min, max, média, mediana...**

summary(diabetes2$Insulin)

# 4.0 CONSTRUÇÃO DO MODELO

**Divisão dos dados em treino e teste - 70% dos dados para treino e 30% dos dados para teste. A função sample.split efetua essa divisão dos dados. O parametro splitRatio recebe o percentual.**

set.seed(123) index = sample.split(diabetes2$Pregnancies, SplitRatio = .70) index

### A variável train receberá 70% dos dados (valores do index=true)

train = subset(diabetes2, index == TRUE)

### A variável test receberá os outro 30% dos dados (valores do index=false)

test = subset(diabetes2, index == FALSE)

### Verificando por medio da função dim o total de linhas e colunas para verificar se a divisão foi feita corretamente.

dim(diabetes2) dim(train) dim(test)

**Tudo correto: 70% de 712 é 498,4.**

## 4.1 TREINANDO A PRIMEIRA VERSÃO DO MODELO: USO DO ALGORITMO KNN

**O algoritmo escolhido para a primeira versão do modelo é o KNN.É um modelo de classificação que se baseia nos vizinhos mais próximos. Quando um novo dado é apresentado, ele irá classificá-lo com base nos exemplos mais próximos apresentados na fase de treinamento**

### Para treinar o modelo, vamos utilizar a função train, do pacote caret

?caret::train

### Treinando a primeira versão do modelo - KNN. A variável resposta é a coluna Outcome

**Nos parâmetros, além da variável resposta, informamos um . após a variável resposta para informar que todas as demais colunas serão utilizadas como variáveis preditoras. Se quisessemos informar apenas algumas colunas como variáveis preditoras, colocaríamos o nome de cada uma individualmente após o til. Para o parâmetro data, informamos o conjunto de dados de treino e no parâmetro method, informamos o algoritmo a ser utilizado, nesse caso, o knn.**

modelo <- train( Outcome ~., data = train, method = "knn")

**Visualizando os resultados do modelo KNN**

modelo$results

**Apresenta os melhores parâmetros (fine tunning do modelo)**

modelo$bestTune

### Análise dos resuldados do KNN: com o K igual a 9, o modelo apresentou acurácia geral de 71,93%.

## 4.2 - SEGUNDA VERSÃO DO MODELO: KNN COM OUTROS VALORES

**Treinando a segunda versão do modelo - testando o comportamento do modelo com outros valores de k**

**Testaremos um valor de K entre 1 e 20. Para isso, no parametro tuneGrid, utilizaremos a função expand.grid, com k recebendo um intervalo entre 1 e 20.**

modelo2 <- train( Outcome ~., data = train, method = "knn", tuneGrid = expand.grid(k = c(1:20)))

**Visualizando os resultados do modelo**

modelo2$results

**Identificando o melhor valor de k**

modelo2$bestTune

**Visualizando a performance do modelo - gráfico de linhas**

plot(modelo2)

**Analise da segunda versão do modelo: com o valor de k = 19, obtivemos a melhor performance do modelo com KNN, com a acurácia igual a 73,07%.**

## 4.3 - TERCEIRA VERSÃO DO MODELO - ALGORITMO NAIVE BAYES

**Naive Bayes: Algoritmo do tipo supervisionado utilizado para classificação. Ele é amplamente utilizado para classificar instâncias em categorias distintas com base em características relevantes. É baseado no Teorema de Bayes e pressupõe a independência condicional das características (daí o termo Naive = ingênuo)**

**Faz o treino do modelo com o algoritmo Naive Bayes**

modelo3 <- train( Outcome ~., data = train, method = "naive_bayes")

**Visualizando os resultados do modelo**

modelo3$results$modelo3$bestTune

**Análise da terceira versão do modelo - utilização do Naive Bayes. Com os parametros laplace =0, useKernel True e Adjust 1, obteve se a acurácia de 76,23%. Portanto, até o momento, é a melhor acurácia alcançada.**

## 4.4 QUARTA VERSÃO DO MODELO - USO DO ALGORITMO RANDOM FOREST

**Random Forest: é um algoritmo de classificação supervisionada que se utiliza de árvore de decisão com nós, folhas e ramos. Os nós são as variáveis, os ramos os valores possíveis de cada nó e as folhas são o valor final de um nó.**

**Treinando o modelo Random Forest**

modelon4 <- train( Outcome ~., data = train, method = "rpart2" )

**Resultado do modelo com Random Forest**

modelon4

**Observamos que com maxdepht = 2, temos a melhor acurácia: 73,10%**

**Verificando a importância das váriaveis para o aprendizado do modelo**

varImp(modelon4$finalModel)

**Resultado: As colunas "Insulin e Blood Pressure" não contribuem muito para o aprendizado do modelo**

**Treinando o modelo sem as colunas "Insulin e BloodPressure" - train[,c(-3,-5)] exclui as colunas**

modelon4_1 <- train( Outcome ~., data = train[,c(-3,-5)], method = "rpart2" ) modelon4_1

**Observamos que retirando as variáveis Insulin e BloodPressure, a acurácia aumenta para 74,76% com o parametro maxdepth igual a 2.**

**Visualizando a arvore de decisão**

plot(modelon4_1$finalModel)text(modelon4_1finalModel)

## 4.5 QUINTA VERSÃO DO MODELO: ALGORITMO SVM com Kernel radial

**O algoritmo SVM é também um algorito de classificação supervisionado. O SVM busca encontrar o hiperparametro que melhor separa duas classes no espaço de características. Esse hiperplano é escolhido de maneira a maximizar a margem (distância) entre os pontos de dados de cada classe mais próximo ao hiperplano, chamados vetores de suporte. Como muitas vezes os dados não são linermente separaveis no espaço original de características, um kernel é utilizado. Ele é uma função que permite mapear os dados para um espaço de dimensão superior, onde a separação é mais eficaz.**

**Faz o treino do modelo com o SVM, utilizando o kernel sigma**

set.seed(100) modelo5 <- train( Outcome ~., data = train, method = "svmRadialSigma" ,preProcess=c("center") )

modelo5$results$modelo5$bestTune

**Resultado: com sigma 0,03494 e c = 0,5, obteve acurácia de 76,48%**

**Avaliação final: o modelo 5, que se utiliza do SVM com kernel Sigma apresentou melhor acurácia: 76,48%**

# 5.0 AVALIAÇÃO DO MODELO DE PREDIÇÃO ESCOLHIDO - VALIDAÇÃO DO MODELO

**Escolhido o modelo 5, que apresentou acurácia com os dados de treino de 76,48%, vamos agora avalia-lo comparando com os dados de teste (os outros 30% dos dados)**

**Avaliando o modelo**

```
?predict
```

**Testando o modelo com os dados de teste**

```
predicoes <- predict(modelo5,test)
```

**Visualizando o resultado das prediçoes do modelo**

```
predicoes
```

```
?caret::confusionMatrix
```

**Criando a confunsion matrix para verificar os resultados do modelo**

```
confusionMatrix(predicoes, test$Outcome)
```

## O RESULTADO DAS PREDIÇÕES JUNTO AOS DADOS DE TESTE MOSTROU UMA ACURÁCIA DE 79,91% DO MODELO

# 6.0 REALIZANDO PREDIÇÕES COM O MODELO TREINADO E VALIDADO

**Realizando predições**

**Criando um dataframe apenas com o registro de um unico paciente para simular a utilização do modelo**

```
novos.dados <- data.frame( Pregnancies = c(3),
Glucose = c(111.50), BloodPressure = c(70), SkinThickness = c(20),
Insulin = c(47.49), BMI = c(30.80),
DiabetesPedigreeFunction = c(0.34), Age = c(28)
)
```

```
novos.dados
```

**Utilizando o modelo para gerar a previsão - passando os dados do paciente**

previsao <- predict(modelo5,novos.dados)

resultado <- ifelse(previsao == 1, “Positivo”,“Negativo”)

**Verificando o resultado da predição do modelo**

print(paste(“Resultado:”,resultado))

# 7.0 VISUALIZAÇÃO DOS RESULTADOS

**Criando o arquivo com os resultados das predições**

write.csv(predicoes,‘resultado.csv’)

**Lendo o arquivo de previsões que foi gerado**

resultado.csv <- read.csv(‘resultado.csv’)

**Alterando o nome das colunas do dataframe**

names(resultado.csv) <- c(‘Indice’,‘Valor previsto’)

**Visualizando o dataframe**

resultado.csv