RECA
REVISTA ELETRÔNICA DE COMPUTAÇÃO APLICADA
VOL. 2 NRO. 2 2021

Desenvolvimento de um Modelo Preditivo em Prol da Identificação de Futuros Casos de Diabetes

Carolina Oliveira Lamarca da Silva
Graduanda em Engenharia de Software – Uni-FACEF
carollamarca.oliveira@gmail.com

Douglas da Silva Costa
Graduando em Engenharia de Software – Uni-FACEF
dougscsilva@gmail.com

Orientador: Prof. Me. Geraldo Henrique Neto
Mestre em Ciências com Ênfase em Informática Médica - FMRP-USP
geraldohenrique@alumni.usp.br

Resumo
Diabetes é uma doença que não tem cura, mas a pré-diabetes sim, por isso buscamos desenvolver um modelo preditivo com mais de 75% de acurácia, a fim de mitigar casos da doença, por meio da prevenção, pois o estudo facilita a identificação de futuros casos em potencial. Este é um trabalho de caráter exploratório e prático, aliado ao embasamento bibliográfico, no qual quatro modelos preditivos foram gerados a partir de uma base de dados de mulheres norte-americanas com idade entre 21 e 81 anos, utilizando a linguagem R, sendo que os melhores resultados foram do *svmRadialSigma* com 76,48% de acurácia, e o segundo foi o *Naive Bayes* com 76,23% de acurácia, alcançando assim o objetivo do estudo.

Palavras-chave: Análise de Dados. Diabetes. Modelo Preditivo.

Abstract
Diabetes is a disease that has no cure, however pre-diabetes does, which is why we sought to build predictive models with an accuracy greater than 75%, in order to mitigate cases of the disease through prevention, since the study facilitates the identification of potential future cases. This is an exploratory and practical work, combined with a bibliographic basis, in which four predictive models were generated from a database of North American women whose age vary from between 21 to 81 years old, using the R language, with the best results being from svmRadialSigma with an accuracy of 76.48% and Naive Bayes coming in second with an accuracy of 76.23%, thus achieving the study's objective.

Keywords: *Data Analysis. Diabetes. Predictive Model.*

1 Introdução

Diabetes é uma doença crônica que atinge homens, mulheres e crianças. O presente artigo tem o objetivo de fazer o estudo de caso de um *dataset* de mulheres norte-americanas, oriundo da plataforma *Kaggle*, uma comunidade on-line de cientistas de dados e profissionais do aprendizado de máquina subsidiária a *Google*, a fim desenvolver um modelo preditivo com mais de 75% de acurácia, porcentagem comumente utilizada em estudos semelhantes na plataforma *Kaggle*, que possa auxiliar o diagnóstico de futuros casos da doença analisando como modelos preditivos

podem auxiliar na prevenção e combate a doenças. No capítulo 2 são apresentados mais detalhes sobre a doença.

O estudo envolveu a busca da melhor acurácia (no mínimo de 75%) de quatro modelos de análise, treinados com *svmRadialSigma, Naive Bayes* e KNN, sendo que dois modelos foram treinados com o KNN, que são apresentados no capítulo 3 e seus resultados no capítulo 4. Nas subseções 2.7, 2.6 e 2.5, respectivamente, são apresentados conceitos e características dos algoritmos.

A linguagem utilizada para a realização da preparação da base, treino e teste dos modelos foi a linguagem R, fazendo uso da *Integrated Development Environment* (IDE) ou Ambiente de Desenvolvimento Integrado RStudio, sendo apresentadas nas subseções 2.2. e 2.8, respectivamente.

Também foram definidos os conceitos e características de modelo preditivo e de mineração de dados nas subseções 2.3 e 2.4, respectivamente, com finalidade de introduzir o leitor ao tema.

2 Referencial Teórico

Nesta seção serão abordados o tema da pesquisa e sua importância, linguagem de programação, e conceitos e técnicas utilizados para o desenvolvimento do projeto.

2.1 Diabetes

De acordo com a Sociedade Brasileira de Diabetes (SBD), diabetes é uma doença crônica na qual o corpo é incapaz de produzir insulina ou incapaz de incorporá-la corretamente ao organismo.

Segundo a SBD, insulina é um hormônio que controla a quantidade de glicose no sangue, sendo ela um recurso que o corpo utiliza como fonte de energia. Diabéticos possuem níveis altos de glicose no sangue, conhecido como hiperglicemia. Os danos ao organismo ocorrem devido a longos períodos com o nível elevado de glicose no sangue.

De forma geral o diabetes pode ser dividido entre diabetes do tipo 1, diabetes do tipo 2, diabetes gestacional e pré-diabetes.

Diabetes tipo 1 ocorre quando o sistema imunológico ataca erroneamente células beta do organismo, levando a pouca ou nenhuma liberação de insulina, o que causa o acúmulo de glicose no sangue. Esta patologia corresponde de 5 a 10% do total de casos.

> Em algumas pessoas, o sistema imunológico ataca equivocadamente as células beta. Logo, pouca ou nenhuma insulina é liberada para o corpo. Como resultado, a glicose fica no sangue, em vez de ser usada como energia. Esse é o processo que caracteriza o Tipo 1 de diabetes, que concentra entre 5 e 10% do total de pessoas com a doença.(SOCIEDADE BRASILEIRA DE DIABETES, s.d., *online*).

Diabetes tipo 2 surge quando o organismo produz de forma insuficiente ou não utiliza adequadamente a insulina produzida. Esta patologia corresponde, em média, a 90% dos casos.

> O Tipo 2 aparece quando o organismo não consegue usar adequadamente a insulina que produz; ou não produz insulina suficiente para controlar a taxa de glicemia.
> Cerca de 90% das pessoas com diabetes têm o Tipo 2. (SOCIEDADE BRASILEIRA DE DIABETES, s.d., *online*).

Durante a gestação, mudanças ocorrem no corpo da mulher, inclusive mudanças hormonais. Em decorrência disso, a produção de insulina aumenta. Porém quando isso não acontece, o nível de glicose no sangue sobe, gerando o desenvolvimento da diabetes gestacional, que além de prejudicar a saúde da mãe, também pode prejudicar a saúde do bebê.

> Durante a gravidez, para permitir o desenvolvimento do bebê, a mulher passa por mudanças em seu equilíbrio hormonal. A placenta, por exemplo, é uma fonte importante de hormônios que reduzem a ação da insulina, responsável pela captação e utilização da glicose pelo corpo. O pâncreas, consequentemente, aumenta a produção de insulina para compensar este quadro.
> Em algumas mulheres, entretanto, este processo não ocorre e elas desenvolvem um quadro de diabetes gestacional, caracterizado pelo aumento do nível de glicose no sangue. (SOCIEDADE BRASILEIRA DE DIABETES, s.d., *online*).

É classificado como pré-diabetes quando o indivíduo possui níveis elevados de glicose no sangue, mas não na quantidade para ser considerado como diabetes. “O termo pré-diabetes é usado quando os níveis de glicose no sangue estão mais altos do que o normal, mas não o suficiente para um diagnóstico de Diabetes Tipo 2” (SOCIEDADE BRASILEIRA DE DIABETES, s.d., *online*).

Segundo a SBD 50% dos pacientes que se encontram nesta fase, desenvolvem a doença. Além disso, esse é o único estágio reversível da doença, por isso um diagnóstico precoce é de suma importância. “As mudanças do estilo de vida são as bases do tratamento do pré-diabetes. Sabendo que é um quadro reversível, a motivação é que o paciente adote novos hábitos saudáveis por toda vida (FERREIRA, 2017, *online*).

A *International Diabetes Federation*, entidade vinculada à Organização das Nações Unidas (ONU), estima que há mais de 463 milhões de pessoas (entre 20 e 79 anos) com a doença.

2.2 Linguagem R

Segundo o The R Project for Statistical Computing, além de linguagem, R é um ambiente computacional para análises estatísticas e desenvolvimento de visualizações gráficas. É um projeto GNU, um acrônimo recursivo para *GNU’s Not Unix* (GNU não é Unix), um sistema operacional Linux lançado em 1986 por Richard Stallman que teve como objetivo principal e contínuo oferecer um sistema compatível com Unix sendo um *software* livre.

R possui vários recursos estatísticos, sendo que seu ponto forte é a qualidade e a facilidade que visualizações gráficas podem ser desenvolvidas. Por estar dentro de uma política de *software* livre, R fornece uma rota para contribuições com a linguagem.

2.3 Modelo Preditivo

Um modelo é uma representação matemática de um objeto ou processo, construídos para simular fenômenos do mundo real como um passo a mais, na esperança de entender com mais clareza o que realmente está acontecendo (BARI *et al.*, 2017, *online*). Já o modelo preditivo é obtido através de um processo de desenvolvimento de uma ferramenta matemática ou modelo combinados com dados, para resolver algum problema, gerando uma predição assertiva (KUNH e JOHNSON, 2013, *online*; BARI *et al.*, 2017, *online*), sendo este gerado através da análise preditiva.

A análise preditiva é a arte e ciência de usar dados para tomar decisões com mais fundamento. Ela nos ajuda a encontrar padrões ocultos e relações nos dados que podem nos ajudar a prever com maior certeza o que pode acontecer no futuro, assim nos promovendo com valiosos *insights* (BARI *et al.*, 2017, *online*). Tendo como foco primário otimizar a acurácia da predição, ou seja, está focado nas chances que algo irá, ou não, acontecer e não entender o porquê de algo acontecer ou não (KUNH e JOHNSON, 2013, *online*). Um objetivo secundário pode ser o de interpretar o modelo e entender o porquê ele funciona (KUNH e JOHNSON, 2013, *online*).

2.4 Mineração de Dados

Mineração de dados, aliada ao estudo, é um meio pelo qual a descoberta acontece gerando novos conhecimentos que contribuem para melhoria de produtos, sistemas, processos, negócios, etc. (SILVA *et al.*, 2016, *online*).

> Encontrar o que era desconhecido, o que estava escondido pode ser entendido como o ato de descobrir. Considerando que as bases de dados são geralmente volumosas e que o conhecimento pode estar implícito, faz-se necessário um trabalho de busca detalhado – associado a um processo analítico, sistemático e, até onde possível, automatizado. (SILVA *et al.*, 2016, *online*).

De maneira simplificada, a mineração de dados pode ser definida como um processo automático ou semiautomático que explora analiticamente grandes bases de dados, por meio do *machine learning* e usando algoritmos sofisticados como *mining tools*, sendo importantes para o embasamento e assimilação de informações importantes, suportando a geração de conhecimento. É uma subárea de conhecimento que faz uso de conceitos vindo da Inteligência Artificial, Aprendizado de Máquina, Estatística e Banco de Dados (SILVA *et al.*, 2016, *online*; BARI *et al.*, 2017, *online*).

Na literatura especializada se encontram diversas taxonomias para caracterizar as tarefas da mineração de dados (SILVA *et al.*, 2016, *online*). Para Fayyad *et al.* (1996; *apud* SILVA *et al.*, 2016, *online*) as tarefas se dividem em dois níveis, preditivas e descritivas. Tarefas preditivas utilizam os valores dos atributos descritivos para predizer valores futuros ou desconhecidos de outros atributos de interesse. Já as descritivas têm como objetivo encontrar padrões que descrevem os dados de forma que o ser humano possa interpretar. No segundo nível essas são especializadas. No conjunto das tarefas preditivas são incluídas a classificação e regressão. Já no conjunto das tarefas descritivas, as especializações, agrupamento, sumarização, modelagem de dependências e detecção de desvios são incluídas.

2.5 KDD (*Knowledge Discovery in Databases*)

A descoberta de conhecimento em base de dados busca encontrar padrões intrínsecos aos dados nele contido para apresentá-los de uma forma para facilitar a assimilação do conhecimento. Estando associado a um processo analítico, sistemático e, até onde possível, automatizado sendo denominado, para efeitos de simplificação, processo de KDD (SILVA *et al.*, 2016, *online*).

KDD (Descoberta de Conhecimento em Bases de Dados), é composto de quatro fases sendo elas: obtenção de dados, pré-processamento, mineração de dados, pós-processamento, sendo executadas na sequência ou não, uma ou mais vezes (SILVA *et al.*, 2016, *online*).

O processo se inicia com a estruturação das bases de dados, que contém dados referentes à área de interesse, a partir dos quais se deseja descobrir algum conhecimento. No pré-processamento os dados são submetidos a uma série de procedimentos que incluem: organização em um repositório único, como em um *Data Warehouse*; tratamento para eliminar instâncias repetidas e/ou valores discrepantes; seleção para a escolha dos exemplares e/ou atributos relevantes a mineração de dados; normalização para que os valores dos atributos estejam em uma mesma escala e sejam considerados com a mesma relevância pelo algoritmo.

Os procedimentos não se limitam a estes. Após os dados serem trabalhados se inicia com a mineração em si, com tarefas como predição, agrupamento, associação, descrição, sumarização e classificação (SILVA *et al.*, 2016, *online*; BARI *et al.*, 2017, *online*). Ao final, os resultados obtidos são validados, avaliados e apresentados em gráficos, tabelas e relatórios estruturados (SILVA *et al.*, 2016, *online*).

2.6 Naive Bayes

Naive Bayes é um algoritmo de classificação de dados baseado na análise probabilística (BARI *et al.*, 2017, *online*). O algoritmo assume que a presença ou não de um atributo em particular em uma classe não tem relação com a presença ou não de outros atributos. Por exemplo, a classificação de um objeto se baseia em atributos como formato, cor e peso. Para um objeto esférico, amarelo e com menos de 60 gramas um resultado que faz sentido é uma bola de tênis. Mesmo que esses atributos dependem um do outro ou na existência de outros atributos, o algoritmo considera que todos esses atributos contribuem de forma independente para a probabilidade desse objeto ser uma bola de tênis (EMC EDUCATION SERVICES, 2015, *online*).

O *Naive Bayes*, assim como a árvore de decisão, é um dos algoritmos mais utilizados para a classificação de dados pelo fato de apresentar bom desempenho em problemas de classificação quando usado tanto com dados categóricos quanto dados numéricos (SILVA *et al.*, 2016, *online*), tendo como transformar valores contínuos em categóricos através de um processo conhecido como *discretization of continuous variables* (EMC EDUCATION SERVICES, 2015, *online*).

Esse algoritmo é usado em e-mail para o filtro de *spams*, usando classificação de texto e também em detecção de fraudes em seguro de veículos (EMC EDUCATION

SERVICES, 2015, *online*). Foi desenvolvido com base no Teorema de Bayes, de Thomas Bayes (SILVA et al., 2016, online), um estatístico inglês.

2.7 Support Vector Machine

O *Support Vector Machine* (SVM), máquina de vetores de suporte, é um algoritmo de classificação de dados que atribui novos elementos de dados para uma das categorias rotuladas (BARI *et al.*, 2017, *online*).

Na maioria dos casos o SVM é classificador binário. Ele assume que os dados em processamento possuem duas possíveis respostas. Em uma outra versão do algoritmo, *multiclass* SVM (SVM multiclasse), expande o SVM para ser usado como um classificador em um conjunto de dados que contém mais de duas classes, agrupamento ou categoria. O SVM foi um sucesso usado em diversas aplicações como reconhecimento de imagens, diagnósticos médicos e análise textual (BARI *et al.*, 2017, *online*).

O SVM usa uma função matemática frequentemente chamada de *kernel function* (função kernel), que é uma função que combina o novo dado com o que melhor representa dos dados de treinamento em função de predizer o rótulo do dado desconhecido (BARI *et al.*, 2017, *online*).

2.8 RStudio

A RStudio foi fundada em 2009 pelo engenheiro de *software* Joseph J. Allaire. ”JJ Allaire é um engenheiro de *software* e empresário que criou uma ampla variedade de produtos, incluindo *ColdFusion Open Live Writer Lose It!* e RStudio.”(RStudio, s.d.b, *online*).

A *Integrated Development Environment* (IDE) ou Ambiente de Desenvolvimento Integrado RStudio, permite o desenvolvimento de projetos utilizando linguagem R, Python, entre outras linguagens *open source*. A missão da RStudio é produzir cada vez mais conhecimento, por meio de um *software* gratuito e de código aberto, que possibilita o desenvolvimento de projetos de ciências de dados, pesquisa científica e comunicação técnica. (RStudio, s.d.a, *online*)

> A missão da RStudio é criar software gratuito e de código aberto para ciência de dados, pesquisa científica e comunicação técnica. Fazemos isso para aumentar a produção e o consumo de conhecimento por todos, independentemente dos meios econômicos, e para facilitar a colaboração e a pesquisa reproduzível, ambas críticas para a integridade e eficácia do trabalho em ciência, educação, governo e indústria.
>
> RStudio também produz RStudio Team, uma plataforma modular de produtos de software comercial que dá às organizações a confiança para adotar R, Python e outros softwares de ciência de dados de código aberto em escala - para o benefício de muitas pessoas, para aproveitar grandes quantidades de dados, para integrar com os sistemas, plataformas e processos corporativos existentes ou em conformidade com as práticas e padrões de segurança - junto com os serviços *online* para facilitar o aprendizado e o uso na web (RStudio, s.d.a, *online*).

3 Resultados do Processo de Análise de Dados

O objetivo da análise de dados é identificar pacientes com alta probabilidade de serem diagnosticados com diabetes, tendo uma acurácia de no mínimo 75%.

3.1 Exploração e Ajustes da Base de Dados

A base de dados utilizada contém dados referentes a pacientes portadoras de diabetes do sexo feminino.

No primeiro momento utilizamos a função *read.csv* responsável por realizar a leitura de um arquivo com a extensão *csv*, *comma-separated values* (valores separados por vírgula), intitulado *diabetes.csv*. Neste processo o conjunto de dados é lido e armazenado em formato de tabela, criando um *data frame* (Figura 1). O *data frame* foi criado com o nome de *diabetes*.

Posteriormente, utilizamos a função *head*, que retorna os primeiros elementos de um vetor, matriz, tabela, *data frame* ou função, para visualizar os primeiros registros do *data frame*.

A Figura 1 ilustra os comandos e o resultado obtido.

Figura 1 - Leitura dos dados

```
> diabetes <- read.csv(file = "C:/diabetes/diabetes.csv")
> head(diabetes)
  Pregnancies Glucose BloodPressure SkinThickness Insulin  BMI DiabetesPedigreeFunction Age Outcome
1           6     148            72            35       0 33.6                    0.627  50       1
2           1      85            66            29       0 26.6                    0.351  31       0
3           8     183            64             0       0 23.3                    0.672  32       1
4           1      89            66            23      94 28.1                    0.167  21       0
5           0     137            40            35     168 43.1                    2.288  33       1
6           5     116            74             0       0 25.6                    0.201  30       0
>
```

Fonte: os autores

Com a função *str* (Figura 2) conseguimos visualizar de forma compacta a estrutura interna de um objeto R, no caso o *data frame* criado, e nele conseguimos visualizar a quantidade de registros (objetos), e os atributos deles que são denominadas de variáveis (*variables*) com seus respectivos tipos de dados e os 10 primeiros registros do *data frame*.

O *data frame diabetes* contém 768 registros e 9 colunas sendo elas *Pregnancies*, *Glucose*, *BloodPresure*, *SkinThickness*, *Insulin*, *Age* e *Outcome* do tipo inteiro, *BMI* e *DiabetesPedigreeFunction* do tipo numérico.

As colunas apresentam informações como: quantidade de gestação, glicose, pressão sanguínea, espessura da pele, insulina, índice de massa corporal (IMC), função de linhagem de diabetes, idade e o resultado, se a pessoa é diabética ou não.

Figura 2 - Utilização da função *str*

```
> str(diabetes)
'data.frame':   768 obs. of  9 variables:
 $ Pregnancies             : int  6 1 8 1 0 5 3 10 2 8 ...
 $ Glucose                 : int  148 85 183 89 137 116 78 115 197 125 ...
 $ BloodPressure           : int  72 66 64 66 40 74 50 0 70 96 ...
 $ SkinThickness           : int  35 29 0 23 35 0 32 0 45 0 ...
 $ Insulin                 : int  0 0 0 94 168 0 88 0 543 0 ...
 $ BMI                     : num  33.6 26.6 23.3 28.1 43.1 25.6 31 35.3 30.5 0 ...
 $ DiabetesPedigreeFunction: num  0.627 0.351 0.672 0.167 2.288 ...
 $ Age                     : int  50 31 32 21 33 30 26 29 53 54 ...
 $ Outcome                 : int  1 0 1 0 1 0 1 0 1 1 ...
>
```

Fonte: os autores

Os dados podem apresentar valores nulos ou ausentes e isso pode comprometer a construção do modelo, dessa forma, utilizamos as funções *is.na* e *colSums*. A função *is.na* cria um novo objeto R, correspondente ao que foi passado como parâmetro no caso um *data frame*, com o tipo *boolean*, onde existir um dado nulo ou ausente a função atribui verdadeiro, caso contrário, falso. Já a função *colSums* realiza a somatória de todos os elementos das colunas de um vetor ou *data frame*. Ao realizar a soma sobre valores booleanos ela considera verdadeiro como 1 e falso como 0. Utilizando as duas funções em conjunto (Figura 3) conseguimos visualizar se existe algum elemento vazio ou ausente. No *data frame diabetes* não houve nenhum caso, visto que, todas as somatórias das colunas resultaram em 0.

Figura 3 - Verificando se existe registros nulos ou ausentes

```
> colSums(is.na(diabetes))
            Pregnancies                 Glucose           BloodPressure           SkinThickness                 Insulin
                      0                       0                       0                       0                       0
                    BMI DiabetesPedigreeFunction                     Age                 Outcome
                      0                       0                       0                       0
```

Fonte: os autores

A função *read.csv* atribui automaticamente um tipo de dados para as colunas ao criar o *data frame*, mas nem sempre atribui o mais adequado. A coluna *Outcome* foi atribuída ao tipo inteiro, mas ela é uma coluna de dados categóricos, onde o número 1 representa mulheres que desenvolveram a diabete e o número 0 mulheres que não desenvolveram. Para fazer essa conversão utilizamos a função *as.factor*, conforme apresentado na Figura 4. A função *as.factor* é usada para converter um vetor em um fator, os termos categóricos e tipo enumerado também são usados como fatores.

Com a coluna *Outcome* convertida pela função *as.factor* podemos visualizar como os dados estão distribuídos com a função *table*, função responsável pela construção de uma tabela com a quantidade de ocorrências de cada nível de um fator, nesse caso 0 e 1, mulher não diabética e mulher diabética, utilizando de classificação cruzada. Conforme apresentado na Figura 4, o *data frame diabetes* tem 500 casos onde a mulher não desenvolveu diabetes e 268 casos onde ela desenvolveu.

Figura 4 - Convertendo e verificando a distribuição dos fatores

```
> diabetes$Outcome <- as.factor(diabetes$Outcome)
> table(diabetes$Outcome)

  0   1
500 268
>
```

Fonte: os autores

Para entender melhor os dados podemos utilizar a função *summary* aplicada exclusivamente ao atributo intitulado de *Insulin*, Figura 5. Essa função tem o objetivo de sumarizar os dados e apresentar como os mesmos estão distribuídos com informações do valor mínimo e valor máximo, da mediana e da média e a separação dos quartis.

Figura 5 - Utilizando a função *summary*

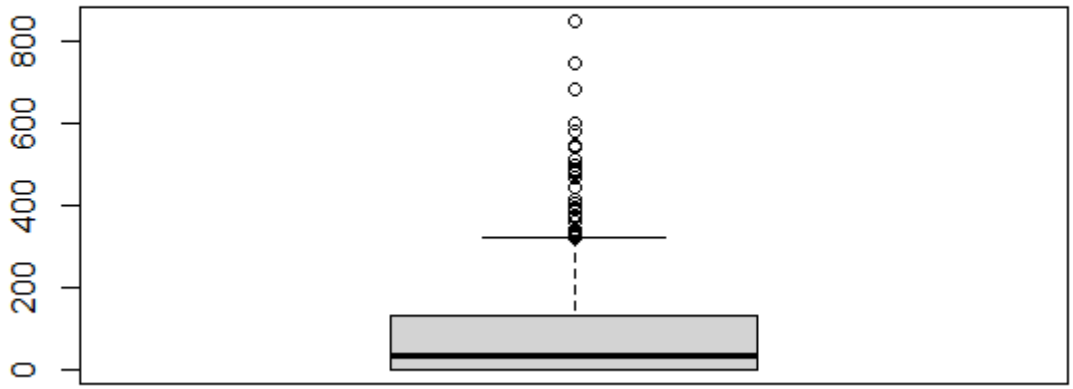
```
> summary(diabetes$Insulin)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    0.0     0.0    30.5    79.8   127.2   846.0
>
```

Fonte: os autores

Uma outra forma de visualizar esses dados é utilizando o *boxplot*, Figura 6. Ele apresenta os dados em formato gráfico, mostrando os valores mínimos e máximos, mediana e os quartis. Além disso, também podemos ver os *outliers*, que são dados discrepantes.

Figura 6 - Gráfico *boxplot* sobre insulina, *data frame diabetes*

```
> boxplot(diabetes$Insulin)
>
```

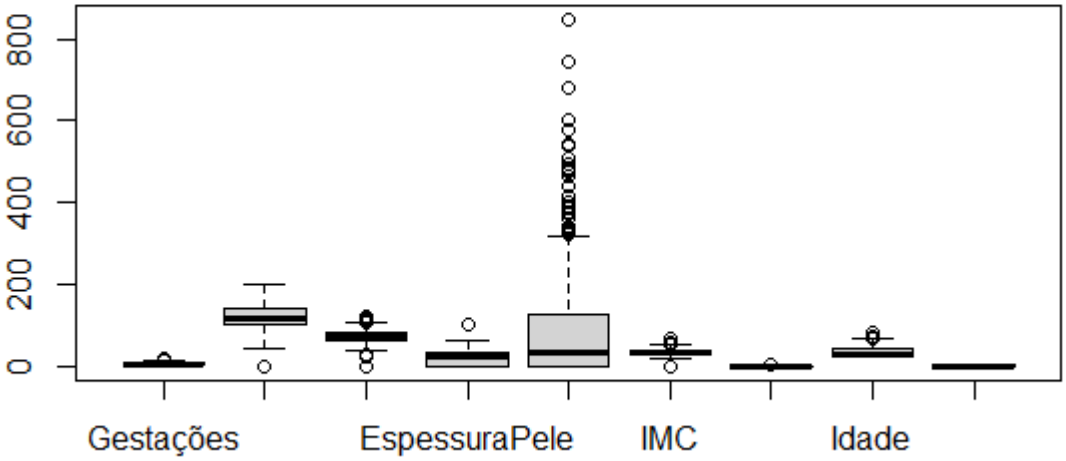


Fonte: os autores

Ao utilizarmos o *data frame diabetes* como parâmetro pudemos verificar os gráficos gerados para todas as colunas, como apresentado na Figura 7.

Figura 7 - Gráfico *boxplot* sobre o *data frame diabetes*

```
> boxplot(
+   diabetes, names=c(
+     "Gestações", "Glicose", "PressãoSanguínea",
+     "EspessuraPele", "Insulina", "IMC",
+     "FunçãoPedigreeDiabetes", "Idade", "Resultado"
+   )
+ )
>
```

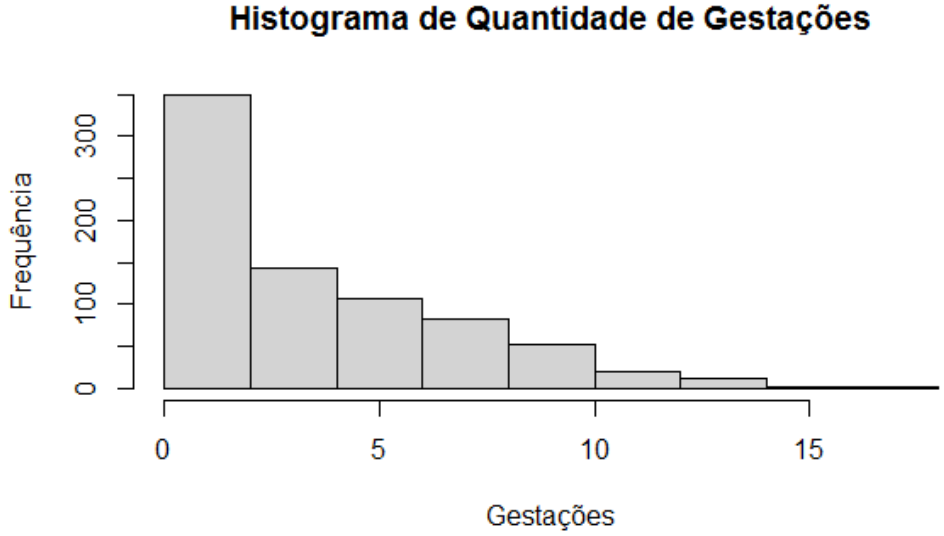


Fonte: os autores

Um outro gráfico que nos auxiliou a interpretar os dados é o histograma. Histograma é um gráfico que nos apresenta a quantidade de ocorrência de um determinado valor. Na Figura 8 temos o histograma que representa a quantidade de gravidez, o histograma de idade (Figura 9), e o histograma de IMC (Figura 10).

Figura 8 - Histograma quantidade de gravidez

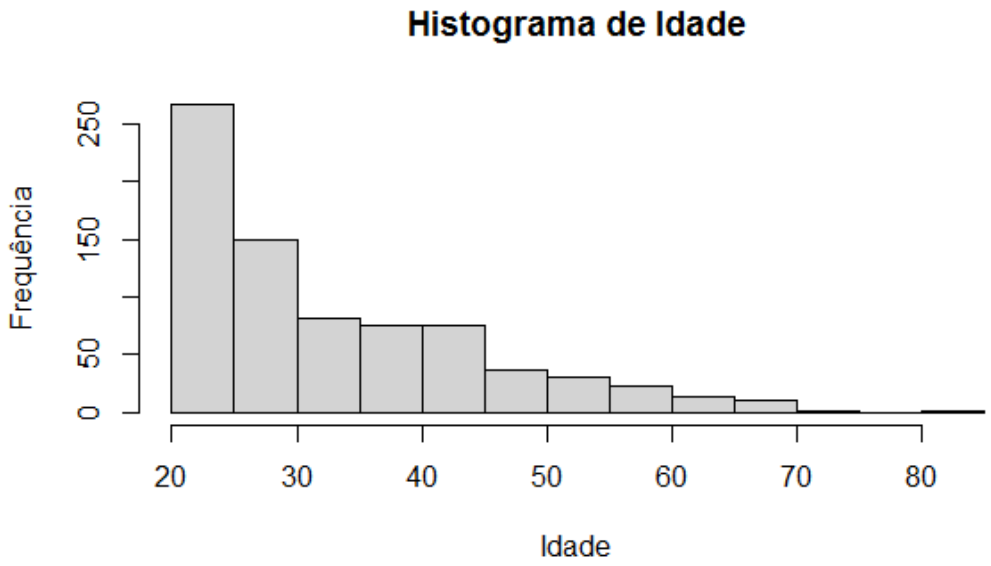
```
> hist(
+   diabetes$Pregnancies,
+   main="Histograma de Quantidade de Gestações",
+   xlab="Gestações",
+   ylab="Frequência"
+ )
>
```



Fonte: os autores

Figura 9 - Histograma idade

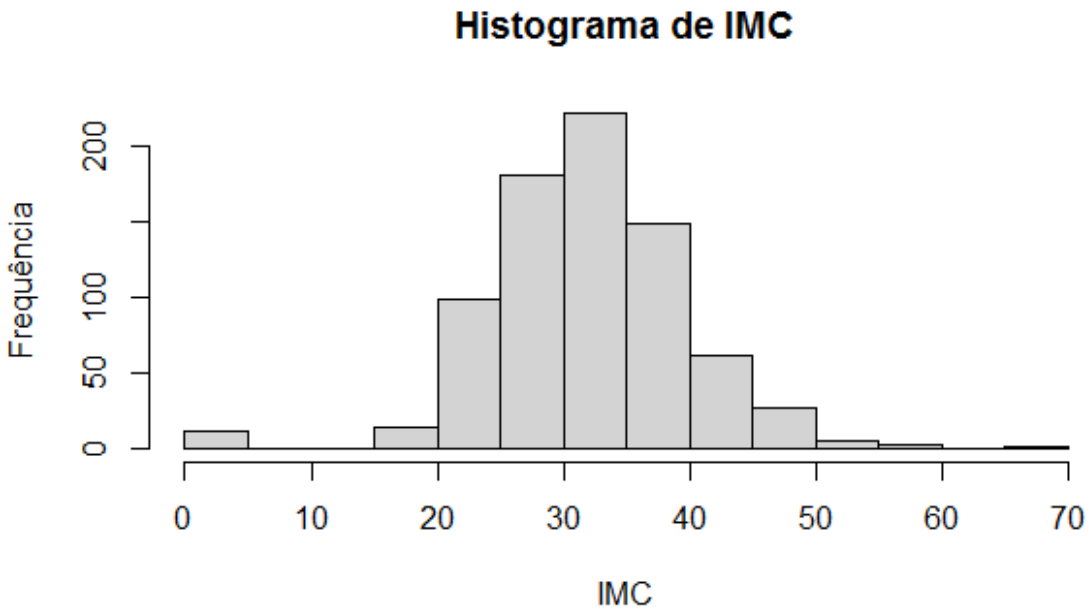
```
> hist(
+   diabetes$Age,
+   main="Histograma de Idade",
+   xlab="Idade",
+   ylab="Frequência"
+ )
>
```



Fonte: os autores

Figura 10 - Histograma IMC

```
> hist(
+   diabetes$BMI,
+   main="Histograma de IMC",
+   xlab="IMC",
+   ylab="Frequência"
+ )
>
```



Fonte: os autores

Assim como os dados nulos e ausentes, os dados discrepantes também podem comprometer a construção do modelo. Para tratar isso utilizamos a função *filter* da biblioteca *dplyr*.

Antes de utilizá-la, devemos instalar e carregar a biblioteca (Figura 11). Utilizamos a função *install.packages* para baixar e instalar a biblioteca e a função *library* para carregar a biblioteca para posterior utilização.

Figura 11 - Instalando biblioteca *dplyr*

```
install.packages("dplyr")
library(dplyr)
```

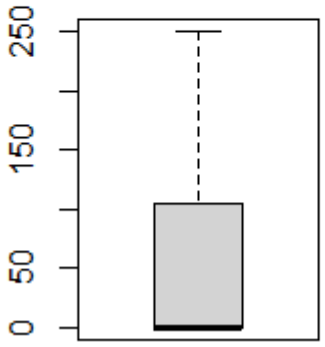
Fonte: os autores

Posteriormente a utilização da função *filter* foi responsável por criar um subconjunto a partir do *data frame* original, contendo todos os registros que satisfazem uma determinada condição de filtragem. A partir desse recurso, foi gerado um novo *data frame*, este nomeado de *diabetes2*, contendo todos os registros onde a insulina é menor ou igual a 250, valor esse utilizado como referência no *kit* utilizado em exames sorológicos.

Podemos visualizar o *boxplot* (Figura 12) e *summary* (Figura 13), que o novo conjunto de dados foi modificado e o *boxplot* agora não apresenta mais nenhum *outlier*.

Figura 12 - Gráfico *boxplot* sobre insulina, *data frame diabetes2*

```
> diabetes2 <- diabetes %>%
+   filter(Insulin <= 250)
> boxplot(diabetes2$Insulin)
>
```



Fonte: os autores

Figura 13 - Utilização da função *summary* nos novos valores de insulina

```
> summary(diabetes2$Insulin)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   0.00    0.00    0.00   55.54  105.00  250.00
>
```

Fonte: os autores

3.2 Construção do Modelo

Com os dados já analisados e preparados, partimos para a construção do modelo.

Utilizamos a biblioteca *caTools* para auxiliar no processo (Figura 14).

Figura 14 - Instalando biblioteca *caTools*

```
install.packages("caTools")
library(caTools)
```

Fonte: os autores

A função *set.seed* é utilizada para configurar uma *seed* para uso no R, *seed* é um vetor inteiro contendo um gerador de números randômicos, *random number generator* (RNG). Ao usar a função passando o mesmo valor como parâmetro as funções que geram números randômicos irão gerar sempre o mesmo resultado, caso o valor do parâmetro for outro os resultados serão distintos.

Utilizando a função *sample.split* da biblioteca *caTools*, ela divide um vetor em dois grupos predefinidos preservando as proporções dos diferentes rótulos, permitindo a criação de um novo vetor do tipo booleano contendo 70% dos elementos como verdadeiro e 30% como falso, conforme passado como parâmetro no *SplitRatio*.

A Figura 15 ilustra os comandos e resultados obtidos.

Figura 15 - Preparando para dividir os *data frame*

```
> set.seed(123)
>
> index = sample.split(diabetes2$Pregnancies, SplitRatio = .70)
>
> index
  [1]  TRUE  TRUE FALSE FALSE  TRUE  TRUE FALSE FALSE  TRUE FALSE  TRUE  TRUE  TRUE
 [14] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
 [27]  TRUE  TRUE FALSE  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE
 [40]  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
 [53]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE FALSE FALSE
 [66]  TRUE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE FALSE
 [79]  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE
 [92]  TRUE FALSE  TRUE FALSE  TRUE  TRUE FALSE  TRUE FALSE FALSE  TRUE  TRUE  TRUE
[105]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE
[118] FALSE FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE
```

Fonte: os autores

Na sequência, utilizamos a função *subset*, função responsável por retornar um subconjunto de uma matriz ou *data frame* que satisfaz uma determinada condição, para criar dois novos *data frames*, um para treino e outro para teste. No *data frame train* foram atribuídos os registros do *data frame diabetes2* no qual o índice corresponde ao índice do vetor *index* onde os elementos são verdadeiros e no *data frame test* onde os elementos são falsos.

A função *dim*, retorna ou define a dimensão de um objeto, para verificar como a atribuição foi realizada. O *data frame* de treino tem 498 registros e 9 colunas, o *data frame* de teste 214 registros e 9 colunas, somando a quantidade de registros dos dois temos os 712 registros do *data frame diabetes2*.

A Figura 16 ilustra os comandos e resultados obtidos.

Figura 16 - Separando e verificando os *data frames* de treino e teste

```
> train = subset(diabetes2, index == TRUE)
> test = subset(diabetes2, index == FALSE)
> dim(diabetes2)
[1] 712   9
> dim(train)
[1] 498   9
> dim(test)
[1] 214   9
>
```

Fonte: os autores

Com os dados segmentados podemos começar o treinamento do modelo, mas antes, torna-se necessário a instalação dos pacotes *caret* e *e1071*, conforme apresentado na Figura 17.

Figura 17 - Instalando bibliotecas *caret* e *e1071*

```
install.packages("caret")
install.packages("e1071")

library(caret)
library(e1071)
```

Fonte: os autores

3.3 Treinamento

Posteriormente, realizamos o treinamento do modelo com a função *train* da biblioteca *caret*. No primeiro parâmetro definimos a variável resposta e as variáveis preditoras separa por ~ (til) neste modelo a variável resposta é a *Outcome* e como utilizamos o . (ponto) todas as outras variáveis serão consideradas preditoras. No parâmetro *data* passamos os dados de treino, nesse caso o *data frame* de treino, e por último no parâmetro *method* o método ao qual o modelo foi treinado, no caso o KNN.

Com o modelo treinado conseguimos acessar algumas propriedades. A propriedade *results* mostra a acurácia para cada valor de k e o *bestTune* o melhor valor de k.

A Figura 18 ilustra os comandos e resultados obtidos.

Figura 18 - Treinando modelo KNN

```
> modelo <- train(
+   Outcome ~., data = train, method = "knn")
> modelo$results
  k  Accuracy     Kappa AccuracySD    KappaSD
1 5 0.6904230 0.2849110 0.03061395 0.05444214
2 7 0.7100934 0.3233486 0.02862777 0.06277927
3 9 0.7193513 0.3377713 0.03149104 0.07021932
> modelo$bestTune
  k
3 9
>
```

Fonte: os autores

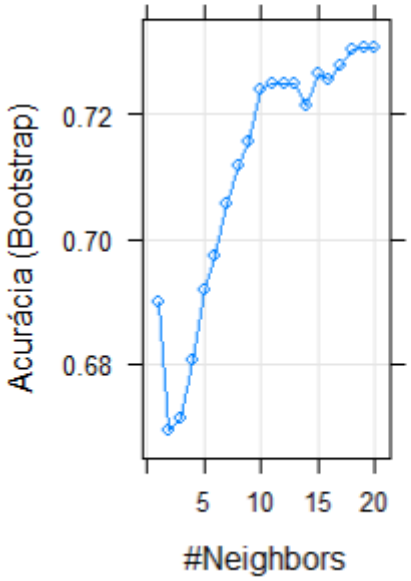
Para testarmos com diferentes valores de k devemos passar o intervalo no parâmetro *tuneGrid*, mas, devemos passar o resultado da função *expand.grid*, que cria um novo *data frame* de todas as combinações dos vetores ou fatores passados como parâmetro.

Para visualizar graficamente usamos a função *plot*, que demonstra a acurácia para cada valor de k.

A Figura 19 ilustra os comandos e o resultado obtido.

Figura 19 - Treinando modelo KNN 2

```
> modelo2 <- train(
+   Outcome ~., data = train, method = "knn",
+   tuneGrid = expand.grid(k = c(1:20)))
> modelo2$bestTune
    k
19 19
> plot(
+   modelo2,
+   ylab="Acurácia (Bootstrap)"
+ )
>
```



Fonte: os autores

Para realizar o treinamento utilizando o método *Naive Bayes* instalamos a biblioteca *naivebayes* (Figura 20).

Figura 20 - Instalando biblioteca *naivebayes*

```
install.packages("naivebayes")
library(naivebayes)
```

Fonte: os autores

Logo na sequência, realizamos o treinamento da mesma maneira, mudando apenas o valor do parâmetro *method* para *naive_bayes*, conforme apresentado na Figura 21.

Figura 21 - Treinando modelo *Naive Bayes*

```
> modelo3 <- train(
+   Outcome ~., data = train, method = "naive_bayes")
> modelo3$results
  usekernel laplace adjust  Accuracy     Kappa AccuracySD    KappaSD
1     FALSE       0      1 0.7501029 0.4192374 0.02737115 0.04705954
2      TRUE       0      1 0.7623195 0.4534239 0.02985181 0.05579072
> modelo3$bestTune
  laplace usekernel adjust
2       0      TRUE      1
```

Fonte: os autores

E por último treinamos com o método *svmRadialSigma*, um SVM utilizado em conjunto com a *Radial Basis Function Kernel*, ou kernel RBF. Para utilizá-lo instalamos a biblioteca *kernlab* (Figura 22).

Figura 22 - Instalando biblioteca *kernlab*

```
install.packages("kernlab")
library(kernlab)
```

Fonte: os autores

Utilizamos o *svmRadialSigma* como valor para o parâmetro *method* e no parâmetro *preProcess*, que recebe um vetor de *string* que define o pré processamento dos dados do preditor, *center*. A Figura 23 ilustra os comandos e resultados obtidos.

Figura 23 - Treinando modelo *svmRadialSigma*

```
> modelo4 <- train(
+   Outcome ~., data = train, method = "svmRadialSigma",
+   preProcess = c("center")
+ )
> modelo4$results
       sigma    C  Accuracy     Kappa AccuracySD    KappaSD
1 0.03494883 0.25 0.7609785 0.4159821 0.02691631 0.05503844
2 0.03494883 0.50 0.7648089 0.4391171 0.02842777 0.05318731
3 0.03494883 1.00 0.7597944 0.4324709 0.03607794 0.07400670
4 0.11527172 0.25 0.7486489 0.3951572 0.03277829 0.06463669
5 0.11527172 0.50 0.7537330 0.4216715 0.03521354 0.07356133
6 0.11527172 1.00 0.7510062 0.4241951 0.03533194 0.07181365
7 0.19559461 0.25 0.7442783 0.3803367 0.03340634 0.06909654
8 0.19559461 0.50 0.7446082 0.4026996 0.03292577 0.06756011
9 0.19559461 1.00 0.7372356 0.3944454 0.02973867 0.06016101
> modelo4$bestTune
       sigma   C
2 0.03494883 0.5
>
```

Fonte: os autores

3.4 Avaliação

Com os modelos treinados passamos para a fase de avaliação. Neste momento avaliamos o desempenho do modelo com dados desconhecidos, os dados de teste do *data frame* pertinente.

Utilizamos a função *predict* (Figura 24), função responsável por predizer o resultado de diversos modelos. Passamos dois parâmetros para a função, primeiro o

modelo, *modelo4* treinado com o método *svmRadialSigma*, e como segundo parâmetro os dados de teste, *data frame test*, e armazenamos as predições em um vetor do tipo fator, como na variável *Outcome*.

Figura 24 - Predições do modelo *svmRadialSigma*

```
> predicoes <- predict(modelo4, test)
> predicoes
  [1] 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0
 [63] 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 1 1 0 0 1 0 1 0 0 1 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 1 0 0 1 0 0 0
[125] 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 1 0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0
[187] 0 0 1 0 0 0 1 1 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0
Levels: 0 1
```

Fonte: os autores

Para analisar com mais detalhes o desempenho utilizamos a função *confusionMatrix* (Figura 25) da biblioteca *caret*, a função retorna a matriz de confusão, matriz que apresenta os acertos e erros na predição, e informações adicionais como a acurácia.

Figura 25 - Matriz de confusão modelo *svmRadialSigma*

```
> confusionMatrix(predicoes, test$Outcome)
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 133  35
         1   8  38

               Accuracy : 0.7991
                 95% CI : (0.739, 0.8506)
    No Information Rate : 0.6589
    P-Value [Acc > NIR] : 4.640e-06

                  Kappa : 0.5092

 Mcnemar's Test P-Value : 7.341e-05

            Sensitivity : 0.9433
            Specificity : 0.5205
         Pos Pred Value : 0.7917
         Neg Pred Value : 0.8261
             Prevalence : 0.6589
         Detection Rate : 0.6215
   Detection Prevalence : 0.7850
      Balanced Accuracy : 0.7319

       'Positive' Class : 0

>
```

Fonte: os autores

Em um outro teste, criamos um novo registro conforme apresentado na Figura 26.

Figura 26 - Criação novo registro

```
> 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
  Pregnancies Glucose BloodPressure SkinThickness Insulin  BMI
1           3   111.5            70            20   47.49 30.8
  DiabetesPedigreeFunction Age
1                     0.34  28
>
```

Fonte: os autores

Posteriormente, utilizamos o objeto na função *predict* no lugar do *data frame*. Também utilizamos as funções *ifelse* e *print*. Com a função *ifelse* pegamos o resultado do fator e transformamos para *string Positivo* ou *Negativo*, no caso o modelo tem como objetivo predizer se uma determinada mulher não é diabética, e a função *print* exibe o resultado no console (Figura 27).

Figura 27 - Mostrando resultado da predição do novo registro

```
> previsao <- predict(modelo4, novos.dados)
> resultado <- ifelse(previsao == 1, "Positivo", "Negativo")
> print(paste("Resultado: ", resultado))
[1] "Resultado:  Negativo"
>
```

Fonte: os autores

4 Resultados

A seguir serão apresentados os resultados dos 4 modelos construídos.

4.1 Resultado dos Modelos

Dentre os 4 modelos criados, o que apresentou melhor acurácia no treinamento foi o *svmRadialSigma* com 76,48% de acurácia, em segundo o *Naive Bayes* com 76,23% de acurácia, em terceiro o segundo modelo KNN treinado com 73,54% de acurácia e com a pior acurácia o primeiro modelo KNN treinado com 71,94% de acurácia.

Assim, apenas os modelos *svmRadialSigma* e *Naive Bayes* cumpriram o objetivo, mínimo de 75% de acurácia.

Conforme os resultados apresentados pela matriz (Figura 25) dos 214 registros utilizados na avaliação, 171 foram classificados corretamente e 43 incorretamente. Dos 171 classificados corretamente, 133 mulheres não diabéticas foram classificadas como não diabéticas e 38 mulheres diabéticas foram classificadas como diabéticas. Dos 43 classificados incorretamente, 35 mulheres diabéticas foram classificadas como não diabéticas e 8 mulheres não diabéticas foram classificadas como diabéticas. A maior taxa de erro refere-se às mulheres diabéticas classificadas como não

diabéticas. Esse erro não é o melhor, pois mulheres com diabetes ou tendência a desenvolver podem não buscar a ajuda de um especialista podendo não tratar ou realizar o tratamento preventivo.

Conforme o resultado apresentado na matriz de confusão, o modelo *svmRadialSigma* performou com uma acurácia de 79,91% para os dados de teste.

Por fim, os resultados da predição do modelo foram armazenados utilizando a função *write.csv*, conforme apresentado na Figura 28, função esta responsável por salvar um objeto em um arquivo *csv*, nesse caso o vetor *predicoes* em um arquivo intitulado de *resultado.csv*, mas antes configuramos o diretório padrão onde o arquivo será salvo com a função *setwd*, função responsável por configurar o diretório padrão para as manipulações de arquivos externos dentro da sessão do RStudio.

Figura 28 - Salvar o resultado em um arquivo *csv*

```
> setwd("c:/temp/")
> write.csv(predicoes, 'resultado.csv')
>
```

Fonte: os autores

5 Conclusão

Concluiu-se que o modelo com maior acurácia foi o *svmRadialSigma* com 76,48%, cumprindo assim o objetivo inicial do estudo, que era encontrar um modelo com uma acurácia mínima de 75%. Além do *svmRadialSigma*, o *Naive Bayes* apresentou 76,23%, também cumprindo o objetivo do estudo. Com isso, esperamos que o presente estudo tenha aplicabilidade a outras bases de dados para que eventuais casos de incidência da doença sejam encontrados e, medidas preventivas sejam alinhadas em prol a uma otimização da saúde pública de um determinado município e ou região.

Uma das maiores dificuldades encontradas foi abstrair os metadados da base de dados, pois os atributos se diferem do habitual, além das informações serem provenientes da área da saúde, canalizando em processos específicos e não corriqueiros. Em relação à linguagem já tínhamos familiaridade pois foi algo abordado em sala de aula, assim como a teoria da utilização dos algoritmos.

Referências

BARI, Anasse; CHAOUCHI, Mohamed; JUNG, Tommy. Predictive Analytics for Dummies. 2. ed. Hoboken: John Wiley & Sons, Inc, 2017.

EMC EDUCATION SERVICES.Data Science & Big Data Analytics: Discovering, Analyzing, Visualizing and Presenting Data. 1. ed. Indianapolis: John Wiley & Sons, Inc, 2015.

FERREIRA, Rita de Cassia Ferreira. Prevenção do Diabetes Mellitus em pacientes pré-diabéticos cadastrados na equipe nova da Estratégia Saúde da Família Paracuri I, Icoaraci, PA. 2017. Disponível em: https://ares.unasus.gov.br/acervo/handle/ARES/9159. Acesso em: 14 out. 2021

KUNH, Max; JOHNSON, Kjell. Applied Predictive Modeling. New York: Springer, 2013.

RStudio, About RStudio. s.d.a. Disponível em: https://www.rstudio.com/about/. Acesso em: 14 out. 2021.

RStudio, J.J. Allaire. s.d.b. Disponível em: https://www.rstudio.com/speakers/j.j.-allaire/. Acesso em: 14 out. 2021.

SILVA, Leandro Augusto da; PERES, Sarajane Marques; BOSCARIOLI, Clodis. Introdução à Mineração de Dados: Com aplicações em R. 1. ed. Rio de Janeiro: Elsevier, 2016.

Sociedade Brasileira de Diabetes.s.d. Disponível em: https://diabetes.org.br/#diabetes . Acesso em: 15 abr. 2021.