MODELO PREDITIVO EFICIENTE PARA ANÁLISE E DETECÇÃO DE PESSOAS COM DIABETES

Alexandre Takaaki Yabuke

RESUMO

O objetivo deste estudo foi o desenvolvimento de um modelo preditivo que pudesse analisar e prever se a pessoa possui Diabetes ou não. A partir de uma base de dados contendo informações relevantes de pacientes, testamos vários algoritmos de Machine Learning e buscamos a obtenção de um modelo preditivo eficiente. Selecionamos alguns algoritmos de classificação como o Support Vector Machine, Dummy Classifier, Logistic Regression, Decision Tree Classifier, K-Neighbors Classifier, Gaussian Naive Bayes, Random Forest Classifier e escolhemos o mais eficiente para a construção do nosso modelo preditivo de classificação. Usamos aleatoriamente 85% da base de dados para fazer o treinamento e os 15% restante para o teste. O pior algoritmo do teste foi o Dummy Classifier e o melhor disparado foi o Random Forest Classifier que atingiu a nossa meta de precisão que era de no mínimo 80%. Apesar da base de dados ser pequena, conseguimos atingir uma precisão bem acima da meta e podemos dizer que foi um excelente resultado. Com uma base de treinamento maior e com mais árvores, o resultado poderia ter sido ainda melhor usando o Random Forest Classifier.

Palavras-chave: Machine Learning. Modelo preditivo. Diabetes.

ABSTRACT

The aim of this study was to develop a predictive model that could analyze and predict whether a person has diabetes or not. From a database containing relevant patient information, we tested several Machine Learning algorithms and sought to obtain an efficient predictive model. We selected some classification algorithms such as Support Vector Machine, Dummy Classifier, Logistic Regression, Decision Tree Classifier, K-Neighbors Classifier, Gaussian Naive Bayes, Random Forest Classifier and chose the most efficient one to build our predictive classification model. We randomly use 85% of the database for training and the remaining 15% for testing. The worst algorithm in the test was the Dummy Classifier and the best was the Random Forest Classifier which reached our accuracy goal of at least 80%. Although the database is small, we managed to reach an accuracy well above the target and we can say that it was an excellent result. With a larger training base and more trees, the result could have been even better using the Random Forest Classifier.

Keywords: Machine Learning. Predictive model. Diabetes.

1. INTRODUÇÃO

Primeiramente vamos ver um pouco do conceito de Inteligência Artificial e Aprendizado de Máquina, assuntos relacionados ao nosso estudo.

Uma inteligência artificial (IA) pode ser definida, de modo amplo, como uma ciência capaz de mimetizar como habilidades humanas e o Aprendizado de Máquina é um subsistema da Inteligência Artificial que mostra a capacidade dos computadores de aprenderem de maneira autônoma, dinâmica e também sem a necessidade de intervenção humana. Responde de acordo com a quantidade de dados que já estão disponíveis para a análise. O Aprendizado de Máquina (Machine Learning) é utilizado para um determinado fim, que se renova e melhora sem ser necessária qualquer intervenção de um programador.

Qual é a importância do Machine Learning hoje? Os crescentes volumes de dados disponíveis, o armazenamento de dados mais acessível e o processamento que é considerado mais poderoso podem interferir diretamente na importância do Machine Learning. Tudo isso que citamos deixa claro que é totalmente possível e necessário reproduzir de maneira ágil, e também automática, modelos que são capazes de analisar tanto dados mais complexos, quanto dados maiores, e o melhor de tudo, com resultados muito mais precisos e eficazes.

O que esperar do Machine Learning na saúde? O Machine Learning pode ajudar o médico a ser mais eficiente, com a mineração e a interpretação de informações dos pacientes. A tecnologia também promete trazer maior precisão aos exames e, em vários países, muitos pesquisadores e cientistas já estão desenvolvendo algoritmos específicos para isso. Essas ferramentas utilizam os bancos de dados disponíveis para encontrar e identificar possíveis alterações potencialmente patológicas. A ideia é que os programas possam aprender com o maior recebimento de subsídios e elementos, mesmo sem a interação humana, fazendo com que eles possam reconhecer padrões. Por fim, outra possibilidade muito interessante da integração entre o Machine Learning e o big data é a associação de dados para diagnósticos. Isso quer dizer que as máquinas e ferramentas que podem aprender e se desenvolver, mesmo sem a influência de um ser humano, para usar o banco de informações disponíveis, fazendo diversas interpretações. Dessa forma, quando um médico digitar

um prontuário, por exemplo, ele estará atualizando o banco de dados em tempo real. Assim, essas informações ficarão disponíveis para outros profissionais ao redor do mundo, o que permite identificar padrões, conhecer condutas, debater tratamentos.

O foco do nosso estudo é exatamente no uso de Machine Learning no diagnóstico de diabetes a partir de determinadas variáveis preditoras.

1.1. Objetivo

O objetivo do nosso estudo é apresentar um modelo preditivo eficiente que indique com certa precisão se um paciente é diabético ou não. A base de dados usada será de 2000 registros e por ser pequena, usaremos 85% dos dados para treinamento e 15% para teste.

Vamos considerar que a meta foi atingida se o modelo preditivo apresentar uma precisão acima de 80%.

2. MODELO PREDITIVO PARA DETECÇÃO DE DIABETES

Para atingirmos o nosso objetivo, utilizamos várias tecnologias que facilitaram o desenvolvimento do nosso estudo. Utilizamos ferramentas IDE que são conjunto de ferramentas básicas necessárias para codificação, debug e teste de software. Também utilizamos a linguagem Python, pois existem diversas bibliotecas prontas para uso relacionadas ao Big Data, Machine Learning, Matemática e Estatística.

2.1. Tecnologias

O Modelo Preditivo foi desenvolvido em linguagem **Python 3.8.5. e bibliotecas Scikit Learn 0.24.2, Matplotlib 3.3.4, Pandas 1.2.5, Numpy 1.20.2. Scipy 1.6.2 e Seaborn 0.11.1.** Utilizamos as ferramentas de desenvolvimento da solução Anaconda com a IDE Spyder.

Python é uma linguagem de programação de alto nível, interpretada de script, imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte. Foi lançada por Guido van Rossum em 1991. Atualmente, possui um modelo de desenvolvimento comunitário, aberto e gerenciado pela organização sem fins lucrativos Python

Software Foundation. Apesar de várias partes da linguagem possuírem padrões e especificações formais, a linguagem, como um todo, não é formalmente especificada.

A Scikit-learn é uma biblioteca de aprendizado de máquina de código aberto para a linguagem de programação Python. Ela inclui vários algoritmos de **classificação**, regressão e agrupamento incluindo máquinas de vetores de suporte, florestas aleatórias, gradient boosting, k-means e DBSCAN, e é projetada para interagir com as bibliotecas Python numéricas e científicas NumPy e SciPy.

Matplotlib é uma biblioteca para criação de gráficos e visualizações de dados em geral, feita para e da linguagem de programação Python e sua extensão de matemática NumPy.

Pandas é uma biblioteca de software criada para a linguagem Python para manipulação e análise de dados. Em particular, oferece estruturas e operações para manipular tabelas numéricas e séries temporais. É software livre sob a licença BSD. O nome é derivado do termo inglês "panel data" (dados em painel), um termo usado em estatística e econometria para conjunto de dados que incluem várias unidades amostrais (indivíduos, empresas, etc) acompanhadas ao longo do tempo.

NumPy é uma biblioteca para a linguagem Python que suporta arrays e matrizes multidimensionais, possuindo uma larga coleção de funções matemáticas para trabalhar com estas estruturas.

SciPy é uma biblioteca Open Source em linguagem Python que foi feita para matemáticos, cientistas e engenheiros. Também tem o nome de uma popular conferência de programação científica com Python. A sua biblioteca central é NumPy que fornece uma manipulação conveniente e rápida de um array N-dimensional. A biblioteca SciPy foi desenvolvida para trabalhar com arrays NumPy, e fornece muitas rotinas amigáveis e bem eficientes como rotinas para integração numérica e otimização.

Seaborn é uma biblioteca de software para criação de gráficos e visualizações de dados baseado no Matplotlib.

Anaconda é uma distribuição das linguagens de programação Python e R para computação científica, que visa simplificar o gerenciamento e implantação de pacotes.

A distribuição inclui pacotes de ciência de dados adequados para Windows, Linux e MacOS.

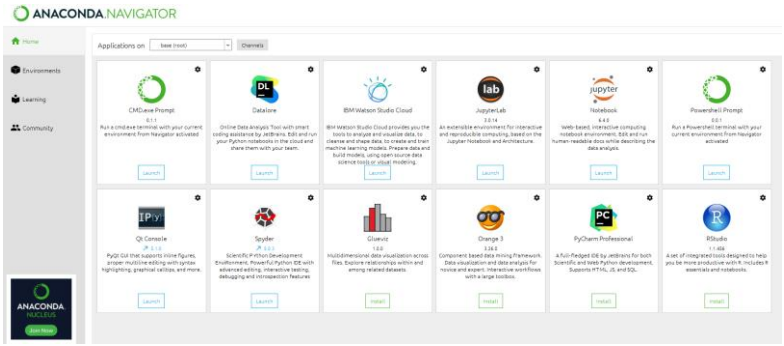


Figura 1 - Anaconda

Spyder é uma IDE multiplataforma de software livre baseada em Python. Seu foco principal é a análise de dados, pesquisa e criação de pacotes científicos, juntamente com o desenvolvimento em Python.

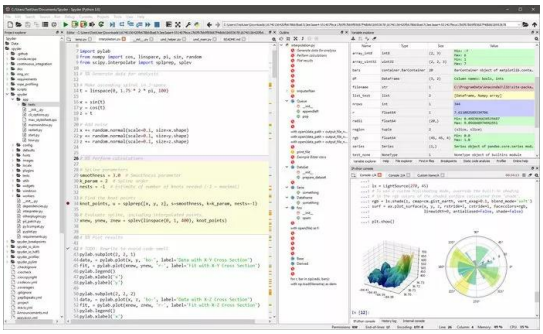


Figura 2 - Spyder

2.2. Preparação do ambiente de desenvolvimento

Primeiramente fizemos o download do Anaconda Individual Edition no site oficial do Anaconda https://www.anaconda.com/products/individual. A instalação foi simples, pois só é preciso iniciar o instalador e seguir as instruções passo a passo até a finalização.

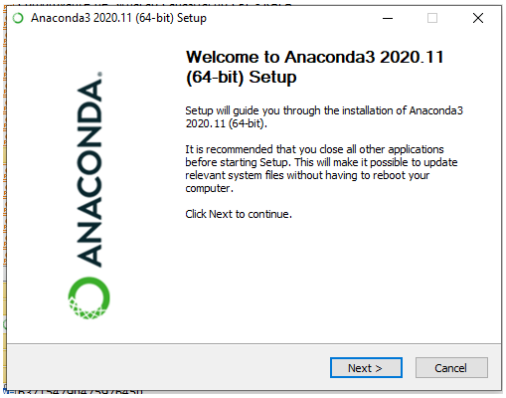


Figura 3 - Instalador do Anaconda

Depois atualizamos as ferramentas e bibliotecas que seriam utilizadas no desenvolvimento, como por exemplo, o próprio Python, Spyder e as todas as bibliotecas Scikit Learn, Matplotlib, Pandas, Numpy e Seaborn. As instruções de como atualizar as bibliotecas podem ser obtidas nos sites oficiais de cada biblioteca. Os links dos sites oficiais são:

ANACONDA	https://www.anaconda.com/
MATPLOTLIB	https://matplotlib.org/
PANDAS	https://pandas.pydata.org/
NUMPY	https://numpy.org/
SCIPY.ORG	https://www.scipy.org/
SCIKIT-LEARN	https://scikit-learn.org/stable/index.html
SEABORN	https://seaborn.pydata.org/

As versões dos pacotes instalados podem ser conferidas no menu Environments do Anaconda.

Name	T	Description	Version
_anaconda_depends		Simplifies package management and deployment of anaconda	2020.07
_ipyw_jlab_nb_ex...		A configuration metapackage for enabling anaconda-bundled jupyter extensions	0.1.0
alabaster		Configurable, python 2+3 compatible sphinx theme.	0.7.12
anaconda		Simplifies package management and deployment of anaconda	custom
anaconda-client		Anaconda.org command line client library	1.8.0
anaconda-project		Tool for encapsulating, running, and reproducing data science projects	0.10.0
anyio		High level compatibility layer for multiple asynchronous event loop implementations on python	2.2.0
appdirs		A small python module for determining appropriate platform-specific dirs.	1.4.4
argh		The natural cli.	0.26.2
argon2-cffi		The secure argon2 password hashing algorithm.	20.1.0
arrow		Better dates & times for python	0.13.1

Figura 4 - Environments

2.3. Dataset

O conjunto de dados público utilizado no nosso estudo pertence ao Instituto Nacional de Diabetes e Doenças Digestivas e Renais da Índia. O objetivo é prever diagnosticamente se um paciente tem diabetes ou não, com base em certas medidas diagnósticas incluídas no conjunto de dados. Todas as pacientes são mulheres com mais de 21 anos de idade.

O conteúdo dos dados consiste em várias variáveis preditoras médicas e uma variável resultado "Outcome". As variáveis preditoras incluem o número de gestações que a paciente teve, índice de massa corporal, nível de insulina, idade e assim por diante.

A tabela abaixo mostra as 8 variáveis preditoras.

Attribute no.	Attribute
1	Number or times pregnant (NTP)
2	Plasma glucose concentration (PGC)
3	Diastolic blood pressure (mmHg) (DBP)
4	Triceps skin-fold thickness (mm) (TSFT)
5	2-h serum insulin (mu U/mL) (H2SI)
6	Body mass index (kg/m2) (BMI)
7	Diabetes pedigree function (DPF)
8	Age

Figura 5- Variáveis preditoras

2.4. Algoritmos de aprendizagem de máquina

Vamos apresentar resumidamente 2 dos principais algoritmos de aprendizado de máquina que utilizamos, Decision Tree (Árvore de decisão) e Random Forest (Floresta aleatória).

2.4.1 Árvore de decisão

O algoritmo Árvores de Decisão é um algoritmo de classificação e regressão para uso em modelagens de previsão de atributos discretos e contínuos. No caso dos atributos discretos, o algoritmo faz previsões fundadas nas relações entre colunas de entrada em um conjunto de dados. Ele usa os valores, conhecidos como estados, dessas colunas para prever os estados de uma coluna que você define como previsível.

No caso de atributos contínuos, o algoritmo usa a regressão linear para determinar onde uma árvore de decisão se divide.

Se mais de uma coluna for definida como previsível, ou se os dados de entrada tiverem uma tabela aninhada configurada como previsível, o algoritmo criará uma árvore de decisão separada para cada coluna previsível.

2.4.2. Floresta aleatória

Floresta Aleatória (Random Forest) é um algoritmo de aprendizagem de máquina fácil de usar que produz excelentes resultados a maioria das vezes, mesmo sem ajuste de hiperparâmetro. É também um dos algoritmos mais utilizados, devido à sua simplicidade e o fato de que pode ser utilizado para tarefas de classificação e também de regressão.

Floresta Aleatória (Random Forest) é um algoritmo de aprendizagem supervisionado, ele cria uma floresta de um modo aleatório. A “floresta” que ele cria é uma combinação (ensemble) de árvores de decisão para obter uma predição com maior acurácia e mais estável. Abaixo, você pode ver uma floresta aleatória com duas árvores:

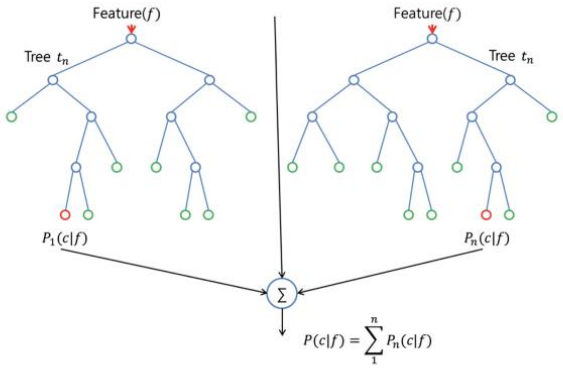


Figura 6 - Floresta aleatória com 2 árvores

Hiperparâmetros importantes

Os parâmetros na Floresta Aleatória são utilizados para aumentar o poder preditivo do modelo ou para tornar o modelo mais rápido.

Podemos aumentar o poder preditivo utilizando o hiperparâmetro que indica o número de árvores construídas pelo algoritmo. Em geral, uma quantidade elevada de árvores aumenta a performance e torna as predições mais estáveis, mas também torna o processamento mais lento e custoso.

Outros hiperparâmetros importantes são os que indicam o número máximo de características a serem utilizadas pela Floresta aleatória na construção de uma dada árvore e o que indica o número mínimo de folhas que devem existir em uma dada árvore.

2.5. Desenvolvimento do Modelo Preditivo para Detecção de Diabetes

A partir de agora, vamos iniciar o desenvolvimento do nosso Modelo Preditivo para detecção de Diabetes.

Primeiramente precisamos abrir o arquivo que contém os dados dos pacientes. O arquivo está em formato CSV e tem o seguinte nome “aty-dataset-diabetes.csv”.

```
In [2]: df = pd.read_csv("./aty-dataset-diabetes.csv")
   ...: df.head()
Out[2]:
   Pregnancies  Glucose  BloodPressure  ...  DiabetesPedigreeFunction  Age  Outcome
0            2      138             62  ...                     0.127   47        1
1            0       84             82  ...                     0.233   23        0
2            0      145              0  ...                     0.630   31        1
3            0      135             68  ...                     0.365   24        1
4            1      139             62  ...                     0.536   21        0

[5 rows x 9 columns]
```

Podemos ver que o arquivo foi lido sem problemas e mostramos as 5 primeiras linhas e as 9 colunas de variáveis. As variáveis são: Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, Diabetes Pedigree Function, Age e Outcome.

A seguir, mostro a quantidade total de registros e o número de diabéticos do nosso conjunto de dados.

```
In [6]: print("[==Dataset==]")
   ...: print(df)
   ...: print("=====")
   ...: print('Quantidade de não Diabetes: ', pd.value_counts(df[labelResultado]) [0])
   ...: print('Quantidade de Diabetes: ', pd.value_counts(df[labelResultado]) [1])
   ...: print("=====")
[==Dataset==]
      Pregnancies  Glucose  ...  Age  Outcome
0               2      138  ...   47        1
1               0       84  ...   23        0
2               0      145  ...   31        1
3               0      135  ...   24        1
4               1      139  ...   21        0
...           ...      ...  ...  ...      ...
1995            2       75  ...   33        0
1996            8      179  ...   36        1
1997            6       85  ...   42        0
1998            0      129  ...   26        1
1999            2       81  ...   25        0

[2000 rows x 9 columns]
=====
Quantidade de não Diabetes:  1316
Quantidade de Diabetes:  684
=====
```

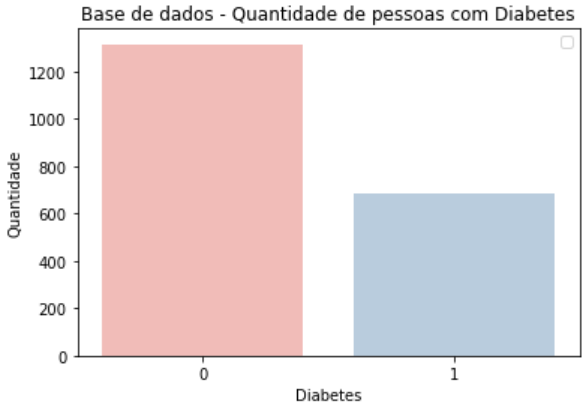
O total de registros é 2000, 684 são diabéticos e 1316 não diabéticos.

Agora vamos conferir se todas as variáveis estão completas ou se teremos que fazer algum tratamento. Abaixo mostro as quantidade de dados para cada variável.

```
In [3]: df.count()
Out[3]:
Pregnancies                 2000
Glucose                     2000
BloodPressure               2000
SkinThickness               2000
Insulin                     2000
BMI                         2000
DiabetesPedigreeFunction    2000
Age                         2000
Outcome                     2000
dtype: int64
```

Podemos perceber que todas as variáveis estão completas, pois o conjunto de dados possui 2000 registros. Se existissem valores nulos, precisaríamos fazer o tratamento de dados. Portanto, não precisaremos fazer nenhum complemento nos dados.

```
In [7]:
   ...: ax = sns.countplot(x=labelResultado, data=df, palette=cor01)
   ...: ax.set_ylabel(xTablela01)
   ...: ax.set_xlabel(yTabela01)
   ...: ax.set_title(titleTabela01)
   ...: ax.legend()
   ...: plt.show()
```



Agora vamos dividir o conjunto de dados em subconjuntos de dados para treinamentos e testes.

Iremos dividir da seguinte maneira: 85% de dados para treinamento e 15% de dados para teste. Como o total de registros é 2000, então teremos 1700 registros para o treinamento e 300 registros para testes. Os dados são divididos aleatoriamente, então não saberemos a quantidade exata de quantos diabéticos e não diabéticos foram divididos para treinamento.

A seguir mostro os Data Frames e Series com os dados divididos para treinamento e teste. A variável porcentagemDadosTeste contém a porcentagem de dados de teste e o restante será para treinamento. As variáveis de saída são X_train (variáveis preditoras de treinamento), X_test (variáveis preditoras de teste), y_train (variáveis de resultado de treinamento) e y_test (variáveis de resultado de teste).

```
In [13]:
    ...: X = df.drop(columns = labelResultado)
    ...: y = df[labelResultado]
    ...: X_train , X_test, y_train, y_test = train_test_split (X, y, test_size =
porcentagemDadosTeste)
```

X_test - DataFrame

Index	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	etesPedigreeFur	Age
279	2	108	62	10	278	25.3	0.881	22
1323	6	104	74	18	156	29.9	0.722	41
1506	0	95	80	45	92	36.5	0.33	26
701	6	125	78	31	0	27.6	0.565	49
819	5	0	80	32	0	41	0.346	37
988	0	107	76	0	0	45.3	0.686	24
454	2	100	54	28	105	37.8	0.498	24
1426	0	102	52	0	0	25.1	0.078	21
309	2	124	68	28	205	32.9	0.875	30
1677	8	151	78	32	210	42.9	0.516	36
1386	4	197	70	39	744	36.7	2.329	31

Figura 7- Variáveis preditoras (Teste)

X_train - DataFrame

Index	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctior	Age
1867	1	146	56	0	0	29.7	0.564	29
1826	2	99	60	17	160	36.6	0.453	21
1333	8	179	72	42	130	32.7	0.719	36
1396	9	164	84	21	0	30.8	0.831	32
664	6	115	60	39	0	33.7	0.245	40
1114	1	120	80	48	200	38.9	1.162	41
1485	0	94	0	0	0	0	0.256	25
399	3	193	70	31	0	34.9	0.241	25
814	8	95	72	0	0	36.8	0.485	57
1118	1	199	76	43	0	42.9	1.394	22
618	9	112	82	24	0	28.2	1.282	50

Figura 8 - Variáveis preditoras (Treinamento)

y_test - Series

Index	Outcome
279	0
1323	1
1506	0
701	1
819	1
988	0
454	0
1426	0
309	1
1677	1
1386	0
1542	0

y_train - Series

Index	Outcome
1867	0
1826	0
1333	1
1396	1
664	1
1114	0
1485	0
399	1
814	0
1118	1
618	1
1633	0

Figura 9 - Variável de resultado (Teste e Treinamento)

O próximo passo foi padronizar todas as variáveis preditoras em uma mesma escala de valores. Como cada variável preditora pode estar em escalas diferentes, precisamos fazer a padronização em uma escala comum e para isso utilizamos o

algoritmo Standard Scaler. O algoritmo aplica em cada variável preditora, valores padronizados de maneira que a média aritmética seja 0 e o desvio padrão seja 1. Se essa fase de padronização não for feita, podemos obter resultados errados. Essa fase é muito importante principalmente quando as variáveis preditoras possuem escalas de dados bem diferentes.

Fórmula

$$M_s = \frac{x_1 + x_2 + x_3 + ... + x_n}{n}$$

Onde,

M_s: média aritmética simples
x_1, x_2, x_3,...,x_n: valores dos dados
n: número de dados

$$DP = \sqrt{\frac{\sum_{i=1}^{n}(x_i - M_A)^2}{n}}$$

Sendo,

∑: símbolo de somatório. Indica que temos que somar todos os termos, desde a primeira posição (i=1) até a posição n
x_i: valor na posição **i** no conjunto de dados
M_A: média aritmética dos dados
n: quantidade de dados

```
In [42]: scaler = StandardScaler()
    ...: X_train_scaled = scaler.fit_transform(X_train)
    ...: X_test_scaled = scaler.transform(X_test)
    ...: print("=====")
    ...: print(X_train_scaled)
    ...: print("=====")
    ...: print(X_test_scaled)
    ...: print("=====")
=====
[[-1.12308277  0.50774174 -0.05239759 ... -0.9126573  -1.02115422
  -1.02437338]
 [ 0.08491515 -1.12542676 -0.57361408 ... -0.53768632 -0.51590493
  -0.42629452]
 [-0.51908381 -0.84276298  0.36457561 ... -0.06272309  0.54728837
  -0.8534937 ]
 ...
 [ 0.68891411  2.32935276  0.05184571 ... -0.15021631 -0.44771177
  -0.16997502]
 [ 1.29291306 -0.43447085  0.5730622  ... -0.93765536  1.18892397
   0.08634449]
 [ 0.08491515  0.35070631  0.8857921  ... -0.51268826 -0.16564008
   2.56409974]]
=====
[[ 2.80291046  0.16226379  1.09427869 ...  1.41216275  0.3427089
   0.76986318]
 [-0.51908381 -0.65432046 -0.26088418 ... -0.3002047  -0.32372422
  -1.02437338]
 [-0.21708433  0.66477717  0.5730622  ...  0.03726917 -0.84447195
   2.56409974]
 ...
 [-0.82108329 -1.56512597 -0.36512748 ... -1.28762827 -0.17493915
  -0.5971742 ]
 [ 0.68891411 -1.28246219  0.5730622  ...  0.96219758 -0.9157648
  -0.42629452]
 [-0.82108329  0.57055591 -1.19907387 ... -0.42519503  0.56278681
  -0.93893354]]
=====
```

Nesse momento já temos todas as variáveis preditoras do subconjunto de dados de teste e treinamento prontas para uso.

O próximo passo é verificar qual é o melhor algoritmo para construirmos o nosso modelo preditivo. Os algoritmos candidatos de classificação são: Support Vector Classification, Gaussian Neive Bayes, Dummy Classifier, Logistic Regression Classifier, Decision Tree Classifier, K-Neighbors Classifier e Random Forest Classifier.

```
In [43]:
    ...: maior = 0
    ...: nomeModelo = ""
    ...: for modelo in [ SVC, GaussianNB, DummyClassifier, LogisticRegression,
    ...:                 DecisionTreeClassifier, KNeighborsClassifier,
    ...:                 RandomForestClassifier,
    ...: ]:
    ...:     clf = modelo()
    ...:     kf = KFold()
    ...:     score = cross_val_score(clf, X_train_scaled, y_train, cv = kf,
scoring=scoringClassification)
    ...:     m = score.mean()
    ...:     print(modelo.__name__, "    ", m)
    ...:     if maior < m:
    ...:        maior = m
    ...:        nomeModelo = modelo.__name__
    ...:
    ...: print("=====")
    ...: print("Melhor modelo:", nomeModelo, "  ", maior )
SVC      0.7997507879958019
GaussianNB      0.6680852527350236
DummyClassifier      0.3447058823529412
LogisticRegression      0.7027413056581755
DecisionTreeClassifier      0.8882377841183405
KNeighborsClassifier      0.7544635767065726
RandomForestClassifier      0.979693851947248
=====
Melhor modelo: RandomForestClassifier    0.979693851947248
```

Olhando para o resultado, podemos perceber que os algoritmos obtiveram pontuação média bem diferentes. O pior resultado foi o Dummy Classifier que teve uma média de 0,3447 e os melhores foram o Random Forest Classifier com média de 0,9797 e Decision Tree Classifier com média de 0,8882.

Portanto, iremos utilizar o classificador Random Forest Classifier no nosso modelo preditivo.

Primeiramente, teremos que escolher o número de árvores do nosso Classificador Random Forest. Quanto maior o número de árvores, melhor o resultado e maior o custo de processamento. Portanto, vamos usar 3000 árvores, pois é um valor que considero razoável.

Depois iremos fazer o treinamento do nosso modelo com os dados de treinamento (X_train_scaled e y_train) e finalmente prever os valores de Y para os dados de teste (X_test_scaled).

```
In [46]:
    ...: rfc = RandomForestClassifier(n_estimators=numeroArvores)
    ...: rfc.fit(X_train_scaled, y_train)
    ...: y_pred = rfc.predict(X_test_scaled)
```

```
In [47]: y_test_array = np.array(y_test.values.tolist())
    ...: print("[Valores corretos]")
    ...: print(y_test_array)
    ...: print("=====")
    ...: print("[Valores preditivos]")
    ...: print(y_pred)
[Valores corretos]
[1 0 0 0 0 1 1 0 1 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 1 1 0 1 0 1
 0 0 1 0 0 1 0 0 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 0 0 1 1 0 0 1 0 1 1 0 0 1
 0 1 0 0 0 0 1 1 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 1 1 1 0
 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 1 1 1 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 1 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 1 0 1 1 1 0 0 1 0 0 0 0 0 1 0 1 0 1 0
 1 0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 1 0 0 1 0 0 0 0 0 1 0 0 0 1
 1 1 1 0 1 0 0 0 0 0 0 1 0 0 0 1 1 1 0 1 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 1
 0 1 0 1 1 0 0 0 0 1 0 0 1 1 0 0 1 1 1 0 1 0 1 0 1 0 0 0 1 0 0 1 1 0 0 0 0
 0 0 0 0]
=====
[Valores preditivos]
[1 0 0 0 0 1 1 0 1 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 1 1 0 1 0 1
 0 0 1 0 0 1 0 0 0 1 1 0 1 0 1 1 1 0 0 1 0 1 1 0 0 0 1 1 0 0 1 0 1 1 0 0 1
 0 1 0 0 0 0 1 1 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 1 1 1 0
 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 1 1 1 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 1 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 1 0 1 1 1 0 0 1 0 0 0 0 0 1 0 1 0 1 0
 1 0 1 0 0 0 1 0 0 0 1 0 0 1 0 1 0 0 0 1 0 1 0 1 0 0 1 0 0 0 0 0 1 0 0 0 1
 1 1 1 0 1 0 0 0 0 0 0 1 0 0 0 1 1 1 0 1 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 1
 0 1 0 1 1 0 0 0 0 1 0 0 1 1 0 0 1 1 1 0 1 0 1 0 1 0 0 0 1 0 0 1 1 0 0 0 0
 0 0 0 0]
```

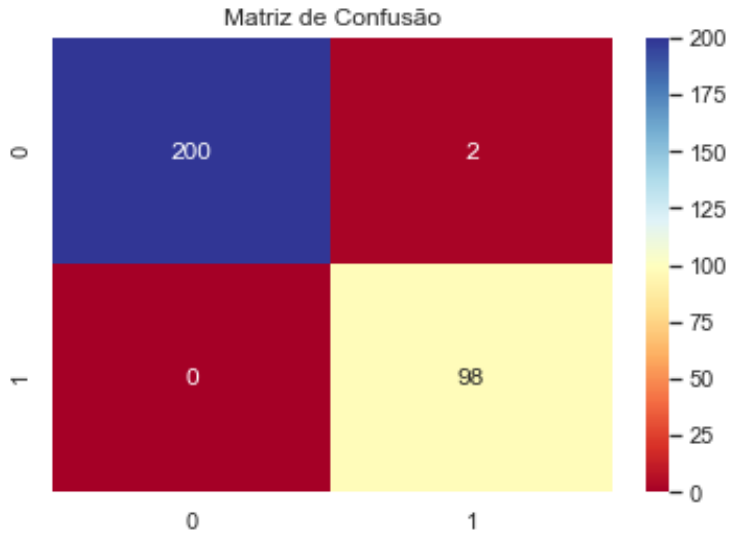
Acima temos dois arrays, o primeiro array contém os valores esperados (0: não diabético e 1: diabético) e o segundo array é a previsão retornada pelo nosso Modelo Preditivo. Podemos perceber que são praticamente idênticas, o que indica que o modelo preditor teve uma boa precisão.

Por último, vamos verificar a precisão de acertos pelo nosso Modelo Preditivo. Lembrando que escolhemos o algoritmo Random Forest Classifier com 3000 árvores e os dados foram divididos aleatoriamente em 85% de dados para treinamento e 15% de dados para teste.

```
In [48]: sns.set()
    ...: ax2 = sns.heatmap(confusion_matrix(y_test, y_pred), annot =True, cmap=cor02,
fmt="d")
    ...: ax2.set_title(titleTabela02)
    ...: plt.show()
    ...: print("=====")
    ...: print(classification_report(y_test, y_pred))
    ...: print ("A precisão de acertos para o conjunto de teste: ",
    ...:        round (accuracy_score(y_test, y_pred) * 100 , 2), "%" )
=====
              precision    recall  f1-score   support

           0       1.00      0.99      1.00       202
           1       0.98      1.00      0.99        98

    accuracy                           0.99       300
   macro avg       0.99      1.00      0.99       300
weighted avg       0.99      0.99      0.99       300

A precisão de acertos para o conjunto de teste:  99.33 %
```

O Modelo Preditivo classificou 300 pacientes e acertou 298 previsões. Acertamos todos os 200 pacientes que não eram diabéticos e 98 pacientes diabéticos. Erramos apenas 2 pacientes que não eram diabéticos e o modelo preditivo indicou erroneamente que era diabético. Portanto, a **precisão foi de 99,33%.**

Abaixo temos a matriz de confusão do resultado para facilitar a visualização dos resultados. As células em vermelho indicam os erros.



Esse foi apenas o primeiro teste, prossegui com mais 9 testes, sempre fazendo aleatoriamente a divisão de teste (15%) e treinamento (85%); e obtivemos os seguintes resultados abaixo.

Teste	Acertos	Erros	Precisão (%)
1	298	2	99,33%
2	295	5	98,33%
3	300	0	100,00%
4	292	8	97,33%
5	298	2	99,33%
6	295	5	98,33%
7	298	2	99,33%
8	293	7	97,67%
9	300	0	100,00%
10	295	5	98,33%
		Média	**98,80%**

Analisando o gráfico, podemos perceber que 2 dos testes obtiveram precisão de 100%. A média de precisão dos 10 testes realizados foi de **98,80%.**

3. Considerações Finais

O que podemos concluir com os resultados ? O nosso modelo preditivo, usando o classificador Random Forest, obteve um ótimo resultado com média de precisão de 98,80% em 10 testes e atingimos a meta de 80%. Outro classificador que também obteve um ótimo resultado foi o Decision Tree Classifier. O único que teve um péssimo resultado em todos os testes foi o Dummy Classifier com média de precisão abaixo de 50%.

O nosso Dataset tinha apenas 2000 registros, por isso foram usados 1700 registros para treinamento. Agora que verificamos que o algoritmo de classificação Random Forest tem um ótimo desempenho para a classificação de diabéticos, o próximo passo para a evolução do modelo preditivo seria uma coleta maior de variáveis preditoras e um número maior de registros. Com um número maior de registros, poderíamos diminuir a porcentagem de dados de treinamento (~70%).

O Dataset utilizado e o código fonte do Modelo Preditivo de Detecção de Diabetes pode ser baixado no link a seguir:

https://drive.google.com/file/d/19YAaBVOVGfnszU9674_DA0aQbmvA_dVd/view?usp=sharing

4. Referências bibliográficas

ALBON, C. **Machine learning with Python cookbook:** Practical solutions from preprocessing to deep learning. O'Reilly Media, 2018.

RASCHKA, S. **Python machine learning.** Packt Publishing, 2015.

HARRISON, M. **Machine Learning – Guia de Referência Rápida**: Trabalhando com dados estruturados em Python. Novatec Editora, 2019

GÉRON, A. **Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow:** Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media, 2019.

MACARAEG, Teodoro **Disease Prediction Model Using Decision Tree Algorithm.** Lap Lambert Academic Publishing, 2021

MULLER, Andreas C. **Introduction to Machine Learning with Python:** A Guide for Data Scientists. 1ª Edição. O'Reilly Media, 2018.

PYTHON.ORG. **Site Oficial do Python,** 2021. Página oficial da linguagem Python. Disponível em: https://www.python.org/. Acesso em 10 de Julho de 2021.

ANACONDA INC. **Site Oficial do Anaconda**, 2021. Página oficial do Anaconda, solução completa para desenvolvimento em Python. Disponível em: https://www.anaconda.com/. Acesso em 10 de julho de 2021.

MATPLOTLIB. **Site Oficial do Matplotlib**, 2021. Página oficial do Matplotlib, biblioteca para criar visualizações gráficas em Python. Disponível em: https://matplotlib.org/. Acesso em 11 de julho de 2021.

PANDAS. **Site Oficial do Pandas,** 2021. Página oficial do Pandas, ferramenta rápida, poderosa, flexível e fácil de usar para análise e manipulação de dados. Disponível em: https://pandas.pydata.org/. Acesso em 11 de julho de 2021.

NUMPY. **Site Oficial do NumPy**, 2021. Página oficial do NumPy, biblioteca fundamental para a computação científica com Python. Disponível em: https://numpy.org/. Acesso em 11 de julho de 2021.

SCIPY.ORG. **Site Oficial do Scipy**, 2021. Página oficial do SciPy, conjunto de bibliotecas em Python para matemática, ciência e engenharia. Disponível em: https://www.scipy.org/. Acesso em 11 de julho de 2021.

SCIKIT-LEARN. **Site Oficial do Scikit-learn,** 2021. Página oficial do Scikit-learn, biblioteca em Python para análise preditiva de dados. Disponível em: https://scikit-learn.org/stable/index.html. Acesso em 12 de julho de 2021.

SEABORN. **Site Oficial do Seaborn**, 2021. Página oficial do Seaborn, biblioteca de visualização de dados baseada em matplotlib. Disponível em: https://seaborn.pydata.org/. Acesso em 12 de julho de 2021.