Centro Federal de Educação Tecnológica de Minas Gerais
Curso de Engenharia de Computação

Uso de aprendizado de máquina para previsão de prováveis pacientes desenvolverem AVC

Gabriela Campos Gama

Orientador: Prof. Dr. Edson Marchetti da Silva
Centro Federal de Educação Tecnológica de Minas Gerais

Belo Horizonte
Julho de 2022

Gabriela Campos Gama

Uso de aprendizado de máquina para previsão de prováveis pacientes desenvolverem AVC

Trabalho de Conclução de Curso apresentado ao Curso de Engenharia de Computação do Centro Federal de Educação Tecnológica de Minas Gerais, como requisito parcial para a obtenção do título de Bacharel em Engenharia de Computação.

Orientador: Prof. Dr. Edson Marchetti da Silva
Centro Federal de Educação Tecnológica de Minas Gerais

Centro Federal de Educação Tecnológica de Minas Gerais
Curso de Engenharia de Computação
Belo Horizonte
Julho de 2022

Resumo

De acordo com a Organização Mundial da Saúde (OMS), o Acidente Vascular Cerebral (AVC) é o segundo maior causador de morte do mundo. O AVC é uma doença grave, e muitas vezes mortal, que pode ocorrer por meio do rompimento ou obstrução dos vasos sanguíneos que levam o sangue ao cérebro. A melhor forma de controlar os fatores de risco, é a adoção de hábitos saudáveis para a prevenção de problemas causados por cada fator de risco. Recursos para classificar o paciente em relação a esses problemas são interessantes para uma melhor compreensão do perfil de cada indivíduo. Diante desse contexto apresentado, a aplicação de técnicas de aprendizado de máquina (AM) pode ajudar, por exemplo, a identificar pacientes com propensão a terem AVC, a partir da realização de predições. Este trabalho tem como objetivo principal desenvolver um modelo, utilizando de técnicas atuais de aprendizado de máquina e disponibilizá-lo em uma página da web. A metodologia do trabalho consiste em aplicar os conceitos de *ensemble*, juntamente com a técnica de generalização em pilha, e treinar diferentes algoritmos, sendo eles *AdaBoost, Random Forest, Extra Trees, Gaussian Naive Bayes, K-Nearest Neighbor* e *Gradient Boosting* produzindo um vetor de predições combinado que é utilizado como entrada para predição com um sétimo algoritmo, *XGBoost*. Avaliando o modelo, obteve-se como resultado uma acurácia de 94,45%, a partir da análise individual dos demais algoritmos, o modelo combinado se mostrou com o desempenho melhor. A utilização de métodos de aprendizado de máquina, com as técnicas utilizadas, como uma ferramenta para a predição de dados de AVC, mostrou-se uma opção promissora e eficaz para predição de AVC. Este trabalho teve como diferencial o desenvolvimento e a disponibilização do modelo em uma página web.

Palavras-chave: Aprendizado de Máquina. Acidente Vascular Cerebral. *Ensemble*. *Generalização em Pilha*.

Abstract

According to the World Health Organization (WHO), stroke is the second leading cause of death in the world. Stroke is a serious and often deadly disease that can occur through the rupture or obstruction of the blood vessels that carry blood to the brain. The best way to control risk factors is to adopt healthy habits to prevent problems caused by each risk factor. Resources to classify the patient in relation to these problems are interesting for a better understanding of the profile of each individual. Given this context, the application of machine learning (MA) techniques can help, for example, to identify patients with a propensity to have a stroke, based on predictions. The main objective of this work is to develop a model, using current machine learning techniques and make it available on a web page. The methodology of the work consists of applying the ensemble concepts, together with the stacked generalization technique, and training different algorithms, namely AdaBoost, Random Forest, Extra Trees, Gaussian Naive Bayes, K-Nearest Neighbor and Gradient Boosting, producing a combined prediction vector that is used as input for prediction with a seventh algorithm, XGBoost. Evaluating the model, an accuracy of 94,45% was obtained, from the individual analysis of the other algorithms, the combined model showed the best performance. The use of machine learning methods, with the techniques used, as a tool for predicting stroke data, proved to be a promising and effective option for stroke prediction. This work had as a differential the development and availability of the model on a web page.

Keywords: Stroke. Machine Learning. Ensemble. Stacked Generalization.

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Lista de Abreviaturas e Siglas

AM	Aprendizado de Máquina
AB	AdaBoost
API	Application Programming Interface
AVC	Acidente Vascular Cerebral
AUC	Area Under the ROC Curve
BMI	Body Mass Index
ET	Extra Trees
GBC	Gradient Boosting Classifier
GNB	Gaussian Naive Bayes
IMC	Índice de Massa Corporal
KNN	K-Nearest Neighbor
OMS	Organização Mundial da Saúde
RF	Random Forest
ROC	Receiver Operating Characteristic
SMOTE	Synthetic Minority Oversampling Technique
XGBoost	Extreme Gradient Boost Classifer

Sumário

1 Introdução

De acordo com a Organização Mundial da Saúde (OMS), o Acidente Vascular Cerebral (AVC) é o segundo maior causador de morte do mundo. A cardiopatia isquêmica e o AVC foram responsáveis por 15,2 milhões de óbitos em 2016. Essas doenças têm sido as maiores causadoras de mortes global pelos últimos 15 anos (OPAS, 2018). No Brasil, em 2017, foram registradas mais de 100 mil mortes causadas pela doença. A Sociedade Brasileira de Cardiologia (SBC) destaca que, entre 1º de janeiro a 16 de outubro de 2020, 78.649 pacientes com AVC foram a óbito (BOND, 2020).

O AVC é uma doença grave, e muitas vezes mortal, que pode ocorrer por meio do rompimento ou obstrução dos vasos sanguíneos que levam o sangue ao cérebro. O AVC hemorrágico, é o mais grave e com maior taxa de mortalidade, ocorre pelo rompimento de um vaso sanguíneo ou uma artéria cerebral, causando o vazamento do sangue. O AVC isquêmico tem maior índice de casos registrados e ocorre devido ao entupimento do vaso sanguíneo pelo acúmulo de gordura ou por um coágulo sanguíneo, impedindo o fluxo de sangue. Os fatores de risco do AVC são classificados de duas formas. A primeira são os fatores modificáveis (aqueles que podem ser controlados): hipertensão, diabetes, tabagismo, sedentarismo entre outros. A segunda forma classifica-se como fatores não modificáveis (aqueles que são imutáveis): sexo, idade, raça, histórico de AVC na família (OLIVEIRA, 2016).

Tendo isso em vista, Abramczuk e Villela (2009) afirmam que o melhor tratamento para o AVC ainda é a prevenção e esse deve ser o foco maior das atenções, tanto dos serviços públicos quanto do setor privado. Essa afirmação também é reforçada no trabalho de Oliveira (2016), reafirmando que a melhor forma de controlar os fatores de risco, é a adoção de hábitos saudáveis para a prevenção de problemas causados por cada fator de risco. O autor ainda adiciona que recursos para classificar o paciente em relação a esses problemas são interessantes para uma melhor compreensão do perfil de cada indivíduo.

Diante da gravidade do contexto apresentado, a aplicação de técnicas de aprendizado de máquina (AM) pode ajudar, por exemplo, a reduzir a ocorrência de AVC, analisando os dados dos pacientes a fim de detectar padrões e fazer previsões. Ou seja, o AM pode auxiliar profissionais de saúde a selecionar os indivíduos de mais alto risco para propor a eles um tratamento preventivo. Portanto, esses modelos preditivos criados a partir de dados provenientes de diferentes fontes, tais como dados demográficos, clínicos e testes laboratoriais, entre outros, podem ser de grande importância para lidar com problemas como o aqui exemplificado (OLIVEIRA, 2016).

Como principais resultados, além de contribuir para auxiliar os profissionais da área atuarem na prevenção e detecção precoce de AVC, e incentivar indivíduos a controlarem os

fatores de risco, este trabalho busca relatar os resultados de porcentagem de acurácia dos modelos desenvolvidos para a predição da doença, de forma a mostrar que há uma relação entre os dados de entradas, referentes aos fatores de risco, como idade, sexo, diabetes, sedentarismo, tabagismo etc, com a ocorrência de derrames cerebrais.

1.1 Contextualização do estudo

Diante do seguinte contexto, no qual o AVC é o segundo maior causador de mortes do mundo, e que o melhor tratamento é o preventivo, o estudo realizado neste trabalho buscou entender a relação dos fatores de risco com as causas do AVC, aplicando os conceitos e técnicas de aprendizado de máquina, de forma a justificar e comprovar que existe uma relação entre esses fatores de risco e que é possível, por meio de um modelo de AM, que os algoritmos aprendam com os exemplos dos dados de pacientes, de forma que quando novos dados de teste, ou seja, dados de novos pacientes sejam usados, o modelo irá predizer se esse indivíduo pode ter ou não um AVC, comprovando assim, a aplicabilidade dessas técnicas de AM para o auxílio do problema de identificar, precocemente, uma pessoa que pode sofrer de um derrame cerebral.

1.2 Objetivos

A prevenção ao AVC é extremamente importante para que as sequelas graves não aconteçam aos pacientes. A fim de auxiliar a detecção de prováveis pacientes desenvolverem um acidente vascular cerebral, este trabalho tem como objetivo principal desenvolver um modelo, utilizando de técnicas atuais de aprendizado de máquina supervisionado, capazes de prever a probabilidade de ocorrer um AVC em um dado paciente, considerando os principais fatores de risco como parâmetros de entrada. Comparar e analisar os resultados dos modelos utilizados no trabalho, de forma a avaliar a precisão dos modelos desenvolvidos. Por fim, disponibilizar o modelo em uma página web para uso com novos exemplos.

1.3 Justificativas e contribuições

Este trabalho justifica-se como um estudo que aplica técnicas de aprendizado de máquina, usando como base de dados informações de paciente reais para treinar os modelos, obtendo resultados que podem contribuir para identificação de possíveis pessoas serem propensas a ocorre ou não um AVC, uma vez que a aplicação das técnicas geram resultados que podem ser utilizados para o auxílio de diagnósticos. Assim o diagnóstico médico será realizado com maior precisão e a patologia detectada em menor tempo (TAKAKURA *et al.*, 2018).

Dessa forma, no âmbito de contribuições, o resultado deste trabalho poderá auxiliar profissionais da área da saúde a identificarem indivíduos propensos a desenvolverem um derrame cerebral, e tomar as devidas medidas para se obter um tratamento bem sucedido. A prevenção é o melhor caminho na luta contra o AVC, portanto, os indivíduos que queiram melhorar a saúde, poderão mudar os hábitos que estão relacionados aos principais fatores de risco, para dessa forma, poder controlá-los e a diminuir o risco de ocorrer a doença.

1.4 Estrutura do trabalho

Este trabalho está estruturado em sete capítulos. Neste primeiro capítulo, a introdução, apresentou-se o contexto, a motivação e objetivos, contextualização do estudo, os resultados esperados e as principais contribuições e justificativas que levaram a elaborar o trabalho.

No capítulo dois, são abordados os principais conceitos relevantes para o entendimento do trabalho, como Aprendizado de Máquina (AM) e Aprendizado de máquina supervisionado.

O capítulo três traz alguns trabalhos relacionados, apresentando semelhanças e diferenças do presente trabalho com outros encontrados na literatura.

O capítulo quatro define e detalha a metodologia utilizada no trabalho.

O capítulo cinco discorre sobre o desenvolvimento para atingir os objetivos propostos.

O capítulo seis detalha os resultados obtidos com análises e inferências.

O capítulo sete apresenta as conclusões do trabalho.

2 Fundamentação Teórica

Neste capítulo será apresentada toda a fundamentação teórica necessária para a compreensão deste estudo. Na primeira seção será discutido o conceito de aprendizado de máquina e aprendizado de máquina supervisionado. Na segunda seção é apresentado uma explicação sobre cada um dos algoritmos utilizados no trabalho. A terceira seção contém a explicação do método de *ensemble*. Na quarta seção é apresentado a generalização em pilha (*stacked generalization*) que é um algoritmo de *ensemble* sofisticado. Por fim, na última seção, é apresentado o conceito de validação cruzada, uma técnica bastante utilizada para avaliação dos modelos de AM.

2.1 Aprendizado de máquina

O Aprendizado de Máquina (AM) é uma subárea da Inteligência Artificial (IA) que estuda métodos capazes de extrair informações de uma base de dados e usá-las para classificar ou prever novos valores. Nesse sentido, uma das possíveis aplicações das técnicas de AM podem ser empregadas na indução, a partir de um conjunto de exemplos de um classificador, pois além de classificar é possível predizer valores também da classe de novos dados quaisquer do domínio em que ele foi treinado (ESTEVES; LORENA; NASCIMENTO, 2009).

Faceli *et al.* (2011) declaram que em AM, os computadores são capazes de aprender com base em experiências passadas. Dessa forma, é usado o princípio da inferência para se obter uma conclusão genérica a partir de um conjunto de exemplos. Os algoritmos de AM aprendem a fazer uma indução de uma função ou hipótese que seja capaz de resolver um problema de um dado conjunto de dados (em inglês - *datasets*).

Existem algumas aplicações das técnicas de AM para soluções de problemas corriqueiros, conforme Faceli *et al.* (2011) citam:

- reconhecimento de palavras faladas;
- predição de taxas de cura de pacientes com diferentes doenças;
- detecção do uso fraudulento de cartões de crédito;
- condução de automóveis de forma autônoma em rodovias;
- ferramentas que jogam gamão e xadrez de forma semelhante aos campeões;
- diagnóstico de câncer por meio da análise de dados de expressão gênica.

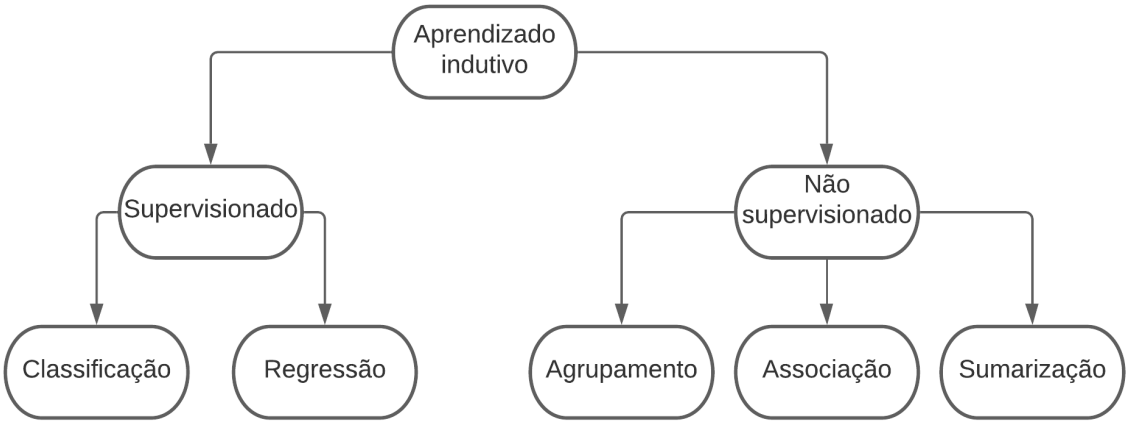
Pode-se classificar em dois tipos diferentes de tarefas que os algoritmos de AM são capazes de resolver, são eles: preditivos e descritivos. Em problemas preditivos, o objetivo principal é encontrar um modelo a partir de um conjunto de dados de treinamento, que seja capaz de prever possíveis saídas para os novos dados. Para isso, cada dado do conjunto de

treinamento deve possuir atributos de entrada e saída conhecidos. Os algoritmos usados nessa tarefa de previsão, seguem o paradigma de aprendizado supervisionado (FACELI *et al.*, 2011).

Já em problemas de descrição, o objetivo é descrever um conjunto de dados, por meio de uma busca de grupos de objetos que sejam semelhantes. Os algoritmos de AM que resolvem esse tipo de problema não fazem uso do atributo de saída, e por isso seguem o paradigma de aprendizado não supervisionado (FACELI *et al.*, 2011).

Faceli *et al.* (2011) descreveram uma hierarquia de aprendizado, que está presente na Figura 1. No topo aparece o aprendizado indutivo, processo que generaliza as soluções. Em seguida, tem-se os tipos de aprendizado supervisionado (preditivo) e não supervisionado (descritivo). Para o tipo supervisionado, os problemas são divididos conforme o tipo de dado, sendo discreto para classificação, e contínuo para regressão. Para o aprendizado não supervisionado, os problemas descritivos são separados em: agrupamentos, de tal forma que os dados são agrupados conforme as semelhanças; associação, que busca encontrar padrões entre os atributos de um conjunto de dados; e sumarização, que tem como objetivo encontrar uma descrição simples para um conjunto de dados.

Figura 1 – Hierarquia de aprendizado



Fonte: adaptado de Faceli *et al.* (2011, p. 6)

2.1.1 Aprendizado de máquina supervisionado

Goodfellow, Bengio e Courville (2016) definem que algoritmos de aprendizado supervisionado são, grosso modo, algoritmos de aprendizado que aprendem a associar alguma entrada a alguma saída, dado um conjunto de treinamento de exemplos de entradas x e saídas y. O termo supervisionado refere-se ao uso de um "supervisor externo" que conhece todas as relações de entradas e saídas. Com isso, o supervisor é capaz de avaliar a capacidade de indução do modelo. Em alguns casos, as saídas podem ser difíceis de se coletar automati-

camente, e precisam ser fornecidas por um humano “supervisor”, mas o termo ainda pode ser aplicado em casos em que a coleta foi realizada automaticamente.

2.2 Algoritmos de classificação

Neste trabalho para lidar com a identificação da propensão ou não de AVC foram escolhidos sete algoritmos de aprendizado supervisionado para resolver o problema de classificação, são eles: *AdaBoost, Random Forests, Extra Trees, Gaussian Naive Bayes, K-Nearest Neighbor, Gradient Boosting e XGBoost*. A razão da escolha desses algoritmos se deve ao fato de serem classificadores bem conhecidos na literatura e bastante utilizados no meio acadêmico para resolver problemas de classificação, como o deste presente trabalho. Nas próximas seções cada um desses classificadores serão dissertados.

2.2.1 AdaBoost

AdaBoost ou *Adaptive Boosting* é um método do tipo *ensemble* que corresponde a uma classe de algoritmos de aprendizagem de máquina que são formados por vários classificadores. A saída do algoritmo produz um resultado combinado da saída dos classificadores que compõem o método. O *ensemble* tipo *Boosting* constrói, na fase de treinamento, hipóteses (modelos de classificadores) sucessivas, de tal modo que exemplos classificados incorretamente por hipóteses anteriores sejam melhor classificados pelas seguintes Freund e Schapire (1996). Após treinado, a saída é obtida combinando todas saídas produzidas por cada hipótese. O algoritmo utilizado para treinar cada modelo de classificador é denominado algoritmo base. Geralmente as árvores de decisão são utilizadas. Freund e Schapire (1996) apresentaram o algoritmo *AdaBoost*, mostrado no algoritmo 1:

Algoritmo 1: O algoritmo *ADABOOST*

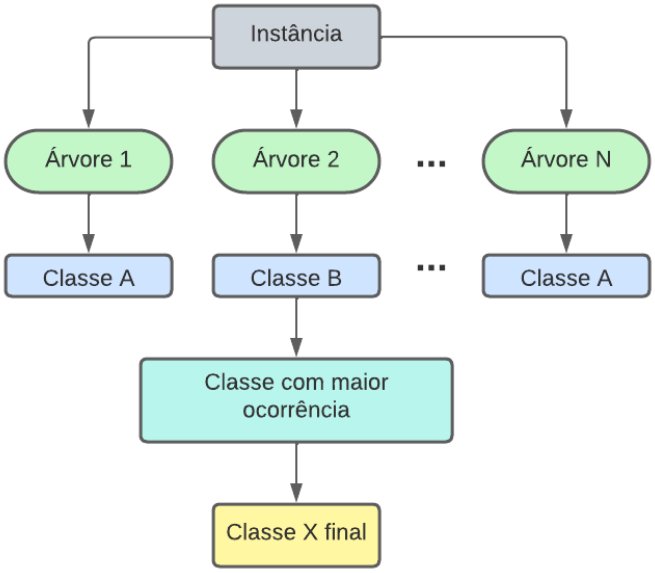
Input: Um algoritmo de aprendizado ϕ
Um conjunto de treinamento $D = \{(x_i, y_i), i = 1,...,n\}$
Número de Iterações Nr
Um conjunto de teste com nt exemplos $T = \{(x_j,?), j = 1,...,nt\}$
Output: Previsões para o conjunto de teste
/* Fase de treinamento */
for $exemplo\, i \in D$ **do**
  $w(i) \leftarrow 1/n$;
end
for $l = 1$ ***to*** Nr **do**
  for $exemplo\, i \in D$ **do**
    $p_l(i) \leftarrow w_l(i)/\sum_i w_l(i)$;
  end
  /* Chamada ao Algoritmo de Aprendizado */;
  $\hat{f}_l \leftarrow \phi(p_l)$;
  /* Calcular o Erro */;
  $e_l = \sum_i p_l(i)[\hat{f}_l(x_i) \neq y_i]$;
  $\beta_l \leftarrow e_l/(1-e_l)$;
  for $exemplo\, i \in D$ **do**
    $w_{l+i}(i) := w_l(i)\beta_l^{1-[\hat{f}_l(x_i)\neq y_i]}$;
  end
end
/* Faça de Teste */;
for $j = 1$ ***to*** nt **do**
  $\hat{y}_j = arg\, max_{y\in Y} \sum_{l=1}^{Nr} (log \frac{1}{\beta_l})[\hat{f}_l(x_j \in T) = y]$;
end
return *Vetor de previsões* $\hat{y}$;

Fonte: adaptado de Faceli *et al.* (2011, p. 147)

2.2.2 Random Forest

Breiman (2001) introduziu o algoritmo das florestas aleatórias (do inglês, *Random Forest*). O modelo gera várias árvores de decisão cujas previsões são combinadas por votação uniforme. Para cada árvore, n atributos são escolhidos aleatoriamente. Após a escolha dos atributos pelo algoritmo, as árvores são construídas utilizando o ganho de informação. Sua estrutura é favorecida pela execução de forma distribuída, tornando-se eficiente para problemas de classificação em grandes bases de dados. A Figura 2 exemplifica o funcionamento do algoritmo *Random Forest*.

Figura 2 – Algoritmo Random Forest



Fonte: adaptado de Machado (2020, p.35)

2.2.3 Extra Trees

O algoritmo *Extra Trees* se assemelha bastante ao *Random Forest*. A principal diferença entre esses dois algoritmos é o fato de que o *Extra Trees* adiciona uma camada a mais de aleatoriedade para construir as árvores. Em vez de utilizar o ganho de informação, ele utiliza uma estratégia aleatória para gerar os nós das árvores. Essa camada a mais de aleatoriedade traz como vantagem a diminuição do viés e o melhor desempenho para resolver problemas mais complexos (GEURTS; ERNST; WEHENKEL, 2006).

2.2.4 Gaussian Naive Bayes

O algoritmo *Naive Bayes* Lewis (1998), utiliza uma abordagem probabilística baseada no Teorema de Bayes, equação (1). O método *Naive Bayes* assume que todas as características são independentes entre si. Ou seja, a presença de uma característica não está relacionada a nenhuma outra, por isso da palavra “Naive (ingênuo)”.

$$p(c \mid x) = \frac{p(x \mid c)p(c)}{p(c)} \tag{1}$$

Dado um elemento $x = x_1, x_2,...,x_n$ com n atributos, o *Naive Bayes* prediz a probabilidade de x pertencente a classe C_k , utilizando a probabilidade calculada pelo Teorema de

Bayes, equação (2):

$$(C_k \mid x) = \frac{p(x \mid C_k)p(C_k)}{p(x)} = \frac{p(x_1,x_2,...,x_n \mid C_k)p(C_k)}{p(x_1,x_2,...,x_n)} \tag{2}$$

Onde C é o conjunto com as k classes possíveis. Após calcular as probabilidades de x pertencente a cada uma das k classes, é atribuído a classe C_k com maior probabilidade ao elemento x.

Geralmente esse método necessita de uma base de treinamento relativamente pequena, e pode ser extremamente rápido se comparado com métodos mais sofisticados, sendo muito utilizado em problemas de previsão em tempo real. Um dos pontos fracos do *Naive Bayes* é justamente sua suposição de independência. Na vida real, é difícil encontrar problemas que sejam completamente independentes.

2.2.5 K-Nearest Neighbor

O algoritmo *k-Nearest Neighbor* (KNN) é um algoritmo de aprendizado supervisionado introduzido por Aha, Kibler e Albert (1991). O método KNN é um "aprendiz preguiçoso" (do inglês, *lazy learning*). A ideia geral desse algoritmo consiste em encontrar os k exemplos rotulados mais próximos do exemplo não classificado e, com base no rótulo desses exemplos mais próximos, é tomada a decisão relativa à classe do exemplo não rotulado.

O desempenho do método KNN é influenciado pela escolha do valor do k. Quando o conjunto de treino possui muitos elementos classificados incorretamente por um "especialista", é preferível utilizar o método KNN com $k = 1$, nos demais casos, com $k > 1$. Entretanto, para determinar o valor de k que o método KNN possui melhor desempenho é necessário realizar experimentos de tentativa e erro com valores distintos para k.

2.2.6 Gradient Boosting

É uma técnica utilizada para problemas de classificação e regressão que consiste na produção de um modelo preditivo baseado num conjunto de modelos preditivos fracos, tipicamente árvores de decisão. O modelo foi originalmente porposto por Friedman (2001).

O algoritmo *Gradient Boosting* consiste em um processo iterativo aditivo. O método inicia com uma previsão constante, cujo valor corresponde à média da variável de resposta na amostra de treinamento ($f_0(x) = y$). A cada iteração, um novo termo é adicionado ao modelo corrente, com o objetivo de reduzir gradualmente o erro de previsão. Assim, as atualizações são calculadas seguindo o sentido inverso do gradiente da função objetivo $\psi(y_i,f(x_i))$, em relação às aproximações correntes, $f(x_i)$. O processo repete-se até que uma determinada condição de parada seja satisfeita, por exemplo, um número máximo de iterações M.

As equações (3) descrevem a estratégia aditiva do algoritmo *Gradient Boosting* para construir o comitê de previsores.

$$\begin{aligned}
&\hat{y}_i^{(0)} = f_0(x_i) = \bar{y} \\
&\hat{y}_i^{(1)} = f_0(x_i) + \eta f_1(x_i) = \hat{y}_i^{(0)} + \eta f_1(x_i) \\
&\hat{y}_i^{(2)} = f_0(x_i) + \eta f_1(x_i) + \eta f_2(x_i) = \hat{y}_i^{(1)} + \eta f_2(x_i) \\
&\ldots \\
&\hat{y}_i^{(M)} = \sum_{m=0}^{M} f_m(x) = \hat{y}_i^{(M-1)} + \eta f_M(x)
\end{aligned} \tag{3}$$

Onde $\hat{y}_i^{(M)}$ representa o valor estimado da variável de resposta para a i-ésima observação da amostra de treinamento, após a m-ésima iteração do algoritmo. O parâmetro η, chamado passo de aprendizado, controla a velocidade de convergência do algoritmo, suavizando as atualizações efetuadas a cada iteração. O comitê de previsão é formado pelas funções $f_m(x)$, que pertencem à uma classe de funções parametrizáveis $f_m(x|\theta_m)$, onde os parâmetros são determinados pelo algoritmo de treinamento dos modelos de aprendizado que compõem o comitê.

As funções $f_m(x|\theta_m)$ são árvores de decisão. Os parâmetros θ_m definem os particionamentos e as constantes de aproximação. Portanto, a cada iteração uma nova árvore de decisão é treinada para ajustar os erros obtidos pelo estado corrente do comitê.

$$\theta_m = \{R_j^{(m)}, c_j^{(m)}\}_{j=1}^{J} = argmin\Psi(r_i^{(m)}, f_m(x_i|\{R_j, c_j\}_{j=1}^{J})) \tag{4}$$

Por fim, o comitê é atualizado com a adição da nova árvore de decisão recém treinada, conforme a equação (5).

$$f_m(x) = f_{m-1}(x) + \eta f_m(x|\theta_m) \tag{5}$$

2.2.7 Extreme Gradient Boost

O *XGBoost* (*Extreme Gradient Boosting*) é um algoritmo de classificação que utiliza o conceito de aprendizado de máquina e é uma melhoria do *Gradient Boosting* de Friedman (2001). O *XGBoost* é recente na literatura, sendo originalmente proposto por Chen e Guestrin (2016), é um algoritmo que utiliza do princípio de arvore de decisão.

Um modelo de árvore de decisão por si só é considerado fraco para fazer uma classificação. Sendo assim, o *XGBoost* utiliza do conceito de *ensemble* ao realizar uma combinação

de várias árvores de decisão, formando um comitê, resultando em um classificador com alto poder preditivo.

O algoritmo utiliza o gradiente para aumentar a construção das árvores reforçadas. Dessa forma, se obtém as pontuações dos recursos, indicando a importância de cada característica para o treinamento. Quanto mais uma característica é utilizada para tomar decisões importantes com as árvores impulsionadas, maior será sua pontuação.

2.3 Modelos Múltiplos Preditivos (*Ensemble*)

A primeira referência a uma combinação de modelos na literatura estatística aparece em Barnard (1963), um trabalho que estuda os dados de passageiros de companhias aéreas.

A ideia principal por trás de qualquer modelo múltiplo é baseada na observação de que diferentes algoritmos de aprendizado exploram:

- Diferentes linguagens de representação;
- Diferentes espaços de procura;
- Diferentes funções de avaliação de hipóteses.

É possível desenvolver um conjunto de classificadores que, trabalhando juntos, obtêm um melhor desempenho do que cada classificador trabalhando individualmente?

Um único algoritmo pode não realizar a previsão perfeita para um certo *dataset*. Essa observação é baseada no teorema "Não há almoço grátis" (do inglês, *No free lunch theorem*) de Wolpert e Macready (1997) e em resultados experimentais do projeto Statlog de Michie *et al.* (1994).

> Os algoritmos de aprendizado de máquina possuem algumas limitações e produzir um modelo com alta acurácia é desafiador. A ideia por trás da combinação de diferentes algoritmos é a de que vários algoritmos combinados produzem um resultado melhor do que apenas um algoritmo sozinho Faceli *et al.* (2011, p. 137).

Supondo que para um determinado problema de classificação, tenha-se disponível um conjunto distinto de algoritmos classificadores, não importando como foram escolhidos e treinados, denominados esse conjunto de classificadores de base. A primeira observação que deve ser feita é a de que todos os classificadores do conjunto têm de fazer previsões de forma independente, ou seja, fazer previsões não correlacionadas.

Dois modelos não cometem erros correlacionados quando ambos classificam uma amostra da classe Y_a como pertencente à classe Y_b, sendo que Y_a é diferente de Y_b. Ali e Pazzani (1997) apresentaram a seguinte definição de erro correlacionado: dado um conjunto de classificadores $F = \{f_1(x),...,f_{nc}(x)\}$, $f_i(x) = y$ denota que o modelo i classificou o exemplo x na classe y. $f(x)$ denota a verdadeira classe de x. A correlação de erro entre dois

classificadores i e j é definida pela equação (6) como a probabilidade que os modelos f_i e f_j têm de cometer o mesmo erro:

$$\phi_{ij} = p(\hat{f}_j(x) = \hat{f}_j(x), \hat{f}_i(x) \neq f(x)) \tag{6}$$

O grau em que os erros em F são correlacionados, e $\phi_e(F)$, possui a seguinte definição conforme equação (7):

$$\phi_e(F) = \frac{1}{nc(nc-1)} \sum_{i=1}^{nc} \sum_{j \neq i}^{nc} \phi_{ij} \tag{7}$$

Onde nc representa o número de modelos no conjunto. Porém essa definição não satisfaz a propriedade de que a correlação entre um objeto e ele mesmo seja de 1. Para incluir essa propriedade a correlação de erro entre pares de classificadores é definida como a probabilidade condicional de dois classificadores cometerem o mesmo erro dado que um deles comete um erro. Essa definição de correlação de erro reside no intervalo [0,1] e a correlação entre um classificador e ele mesmo é 1. A definição formal é representada pela equação (8):

$$\phi_{ij} = p(\hat{f}_i(x) = \hat{f}_j(x) \mid \hat{f}_i(x) \neq f(x) \vee \hat{f}_j(x) \neq f(x)) \tag{8}$$

Levando em consideração que ϕ_{ij} é simétrico a equação (8) pode ser reescrita na equação (9):

$$\phi_e(F) = \frac{2}{nc(nc-1)} \sum_{i=1}^{nc} \sum_{j>1}^{nc} \phi_{ij} \tag{9}$$

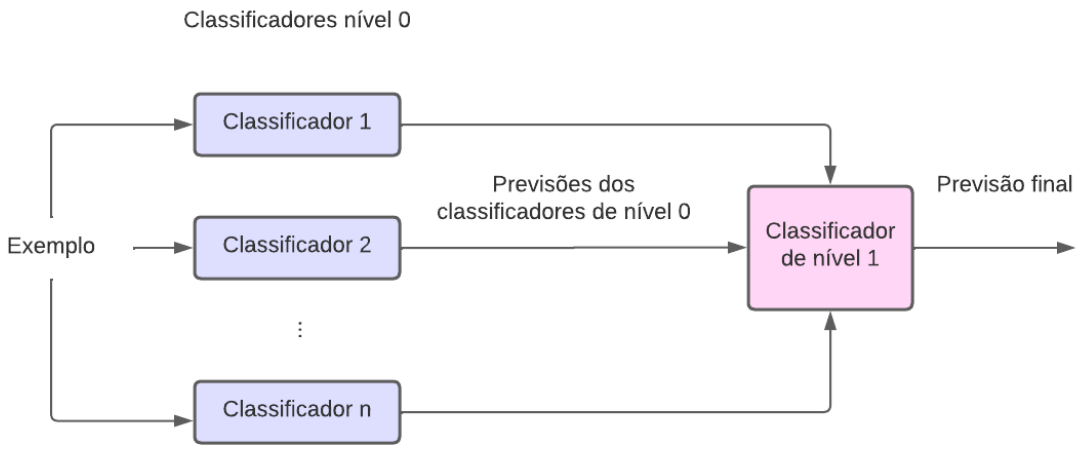
2.4 Generalização em Pilha (*Stacked Generalization*)

O método Generalização em Pilha (do inglês, *Stacked Generalization* ou *Staking*), foi proposto por Wolpert (1992), tendo como característica o aprendizado estruturado em camadas. Os algoritmos classificadores do nível 0 recebem como entrada os dados originais, e cada classificador faz uma predição. As próximas camadas recebem como entrada as predições dos classificadores anteriores, e produzem uma nova predição, passando assim sucessivamente para a próxima camada. Um único classificador no nível mais alto produz a predição final.

Generalização em Pilha é um processo para minimizar o erro de generalização usando classificadores nas camadas mais altas para aprender o tipo de

> erro cometido pelo classificador imediatamente abaixo. Nessa perspectiva, ela pode ser vista como uma extensão para métodos de seleção, nomeados validação cruzada, que usa uma estratégia de "o vencedor leva tudo". Um único classificador com erro de validação cruzada mais baixo é selecionado. A ideia por trás da generalização em pilha é que pode haver um modo mais inteligente para usar o conjunto de classificadores. A regra dos classificadores dos níveis mais altos é aprender como os classificadores anteriores cometem erros, em qual classe eles concordam ou discordam, e usar o seu conhecimento para fazer predições Faceli *et al.* (2011, p. 151).

Figura 3 – Arquitetura da generalização em pilha



Fonte: adaptado de Faceli *et al.* (2011, p. 151)

A Figura 3 representa a arquitetura das camadas do método de *stacking*. A Tabela 1 mostra o conjunto de dados original e o conjunto de dados de Nível 1. Nesse último nível, cada coluna designada por P_{ik} representa a probabilidade de o exemplo ser da classe k dada pelo classificador i. Há duas fases distintas que são chamadas de fase de treinamento ou aprendizado e fase de aplicação. A primeira fase consiste em:

Tabela 1 – Exemplo ilustrativo de generalização em pilha. A tabela mostra o conjunto de dados e o conjunto de dados de Nível 1.

V_1	V_2	V_3	V_4	V_5	**Classe**	P_{11}	P_{12}	P_{21}	P_{22}	P_{31}	P_{32}	**Classe**
t	a	c	t	a	Membro	0,51	0,49	0,13	0,87	0,12	0,88	Membro
t	g	c	t	a	Membro	0,19	0,81	0,07	0,93	0,81	0,19	Membro
g	t	a	c	t	Não membro	0,68	0,32	0,55	0,45	0,69	0,31	Não membro
a	a	t	t	g	Membro	0,74	0,26	0,66	0,34	0,94	0,06	Membro
t	c	g	a	t	Não membro	0,62	0,38	0,01	0,99	0,78	0,22	Não membro
a	g	g	g	g	Membro	0,65	0,35	0,90	0,10	0,55	0,45	Membro
Conjunto de dados original						Conjunto de dados de Nível 1						

Fonte: adaptado de Faceli *et al.* (2011, p. 151)

1) Treinar cada um dos classificadores de nível 0 usando validação cruzada, funcionando da seguinte forma: para cada conjunto de exemplos que serão treinados, um exemplo é retirado e os demais são treinados. Ao fim do treinamento, o exemplo que ficou de fora é classificado. Um vetor com todas as predições dos algoritmos classificadores é criado.

2) O classificador do nível 1 é treinado, usando como conjunto de treinamento o vetor criado pelos classificadores do nível 0. Na segunda fase, chamada de fase de aplicação, quando um novo exemplo é apresentado, todos os classificadores do nível 0 produzem uma predição. O vetor de predição é então classificado pelo algoritmo classificador do nível 1, que produz uma única saída com a predição final para o exemplo mostrado na Figura 3.

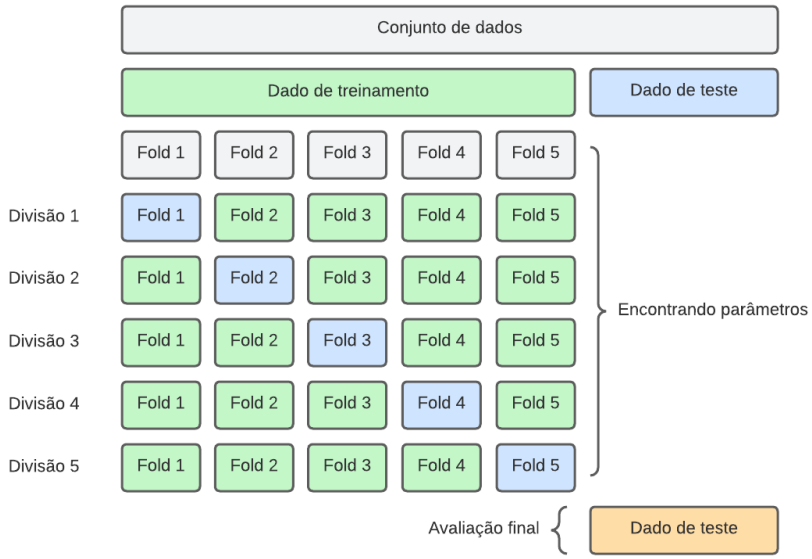
Conforme afirma Breiman (1997), o método de generalização em pilha é uma técnica sofisticada que tem como vantagem a redução de erro devido a redução do viés.

2.5 Validação Cruzada

A Validação Cruzada (do inglês, *Cross Validation*) é uma técnica de avaliação de desempenho de modelos de aprendizado de máquina. A validação cruzada consiste em particionar os dados em subconjuntos (partes), onde um subconjunto é utilizado para treino e outro subconjunto é utilizado para teste e avaliação do desempenho do modelo.

Um dos métodos mais usados de validação cruzada é chamado de *k-fold cross validation*. A técnica consiste em dividir a base de dados de forma aleatória em k subconjuntos, sendo que k é definido previamente, com aproximadamente a mesma quantidade de amostras. A cada iteração de treino e teste, um subconjunto de $k-1$ é utilizado para treinamento, enquanto que o subconjunto restante é utilizado para teste, gerando um resultado de métrica para a avaliação de acurácia. Esse processo garante que cada subconjunto será utilizado para teste. A Figura 4 mostra essa iteração.

Figura 4 – Validação cruzada para k = 5



Fonte: adaptado de Pedregosa *et al.* (2011)

O valor de k deve ser escolhido cuidadosamente de forma que cada conjunto de dados seja grande o suficiente para representarem o conjunto de dados original. Na literatura é comum encontrar a orientação de se utilizar um valor de $k = 5$ ou $k = 10$, mas não é uma regra formal. Um valor de k mal escolhido pode resultar em uma interpretação errada do desempenho do modelo.

3 Trabalhos Relacionados

Na literatura, há diversos estudos que utilizaram diferentes abordagens de aprendizado de máquina para a predição de acidente vascular cerebral. Neste capítulo, serão discutidas algumas publicações que apresentaram técnicas semelhantes ao que este trabalho propõe, de forma a ressaltar o estado da arte a respeito do tema abordado. Todos os trabalhos apresentados foram retirados do repositório IEEE *Xplore*, no qual buscou-se pelas seguintes palavras-chave: "*stroke prediction*", "*machine learning*" e "*artificial intelligence*". Diversos artigos foram encontrados, mas após a leitura dos resumos e verificação dos artigos que possuíam o mesmo escopo deste trabalho, cinco artigos foram escolhidos para serem citados e apresentados a seguir.

Kansadub *et al.* (2015) propuseram um modelo para previsão de AVC com base nos dados demográficos dos pacientes. O estudo comparou a precisão, falso positivo (FP), falso negativo (FN), e a área sob a curva ROC [1] (do inglês, *Area Under the ROC Curve (AUC)*[2]) entre os três modelos desenvolvidos. Com base no princípio da mineração de dados, a pesquisa é dividida em seis processos: coleta de dados, atributos e seleção de dados, pré-processamento de dados, modelagem de dados e avaliação. Os dados demográficos foram coletados da Faculdade de Fisioterapia, Mahidol University, Tailândia, de 2012 a 2015. Depois que os dados estavam prontos, os autores criaram modelos usando três métodos, compreendendo Árvores de Decisão, Naive Bayes e Rede Neural Artificial.

Como resultado, os autores constataram que a Árvore de Decisão provou ter a melhor precisão e menor número de FP, sendo que a baixa taxa de FP significa alta precisão na previsão dos pacientes que tiveram derrame, mas que realmente não o tiveram, enquanto que FN representa a predição incorreta, mas o paciente na verdade teve derrame. Nesse estudo, no aspecto da precisão, a Árvore de Decisão foi o melhor método, mas no aspecto da segurança da vida, a Rede Neural foi o melhor método por causa do maior valor de FP e o menor valor de FN. Por fim, Kansadub *et al.* (2015) concluíram que o estudo teve como benefício encontrar o melhor modelo de previsão de AVC pelo melhor método em dois aspectos: precisão de previsão e segurança de vida, a partir da extração dos dados demográficos dos pacientes.

A pesquisa de Jeena e Kumar (2016), investiga os vários parâmetros fisiológicos que são usados como fatores de risco para a previsão de acidente vascular cerebral. Uma *Support Vector Machine* (SVM) com diferentes funções de *kernel* foram aplicadas em 300 amostras

[1] Receiver Operating Characteristic (ROC) é uma curva de probabilidade. Ela é criada traçando a taxa verdadeiro-positivo contra a taxa de falsos-positivos.

[2] O AUC representa o grau ou medida de separabilidade. Quanto maior o AUC, melhor o modelo está em prever.

de treinamento e testadas com 50 amostras. Função polinomial, quadrática, base radial e funções lineares foram aplicadas e todas obtiveram diferentes precisões. Posteriormente os autores compararam a precisão de classificação das diferentes funções de *kernel*.

Jeena e Kumar (2016) obtiveram como resultado uma precisão de 91% para o modelo com a função linear. Os autores concluíram que o modelo pode ajudar os médicos a planejar a melhor medicação e podem fornecer aos pacientes um diagnóstico precoce de AVC. Ainda ressalvam que o trabalho demonstra o poder preditivo do SVM com um conjunto de parâmetros nos dados de entrada. O método pode ser estendido para um grande banco de dados, considerando mais atributos de entrada, a fim de melhorar o sistema de atuação.

Singh e Choudhary (2017) compararam diferentes métodos de Inteligência Artificial com a abordagem por eles proposta para previsão de AVC, utilizando um conjunto de dados do *Cardiovascular Health Study* (CHS). A metodologia usada se dividiu nas seguintes etapas: coleta de dados, pré-processamento, seleção de recursos, redução de dimensão, classificação e análise dos resultados. Os autores usaram o algoritmo de Árvore de Decisão para o processo de seleção de recursos, o algoritmo de *Principle Component Analysis* (PCA) para reduzir a dimensão, e o algoritmo de *Backpropagation* de Rede Neural Artificial (RNA) , para construir um modelo de classificação. Depois de analisar e comparar a eficiência de classificação e a precisão do modelo proposto com outros diferentes modelos relacionados conhecidos, os autores concluíram que o modelo desenvolvido, apresentou melhor performance de previsão, com precisão de 97,7%.

Emon *et al.* (2020) utilizam diferentes abordagens de aprendizado de máquina para previsão de doenças de AVC. Nessa pesquisa, considerou-se informações de ocorrência de hipertensão, nível de índice de massa corporal, doença de coração, nível médio de glicose, tabagismo, derrame anterior e idade, retirados de uma clínica médica em Bangladesh, totalizando 5.110 pacientes. Usando esses atributos, dez classificadores foram treinados, são eles: *Logistics Regression, Stochastic Gradient Descent, Decision Tree Classifier, AdaBoost Classifier, Gaussian Classifier, Quadratic Discriminant Analysis, Multi layer Perceptron Classifier, K-Neighbors Classifier, Gradient Boosting Classifier, e XGBoost Classifier* para prever o derrame. Posteriormente, os autores agregaram os resultados dos classificadores por meio de um método de *ensamble* com abordagem de votação ponderada para alcançar maior precisão.

No modelo final, Emon *et al.* (2020) obtiveram uma precisão de 97%, em que o classificador de votação ponderada teve melhor desempenho do que os classificadores básicos. A área sob o valor da curva do classificador de votação ponderada também foi alta. Taxa de falsos positivos e taxa de falsos negativos do classificador ponderado foi o mais baixo em comparação com outros. Os autores concluíram que o modelo final de classificador de votação ponderada obteve bons resultados para prever o AVC, podendo ser usado por médicos e pacientes para prescrever e detectar precocemente um possível AVC.

No estudo de Peng *et al.* (2020), os dados fisiológicos dos pacientes foram usados

para construir um modelo de previsão do AVC, com uso de Redes Neurais Artificiais (RNAs). Foi utilizado um conjunto de dados como uma amostra de treinamento e dividido aleatoriamente o conjunto de dados em três subconjuntos. O modelo considerado possui oito arquiteturas e dois algoritmos de aprendizagem, e o número de nós ocultos aplicados são, respectivamente, 1, 2, 5, 10, 15, 20, 25 e 50, enquanto os dois algoritmos de treinamento são *Levenberg–Marquardt* (LM) e *Scaled Conjugate Gradient* (SCG). Cada arquitetura de RNA foi executada 1.000 vezes, com 10.000 épocas. Os resultados simulados indicaram que tanto o algoritmo SCG quanto o LM conseguiram atingir uma taxa média de precisão de classificação acima de 98%, por validação cruzada. Mas, no caso de apenas 1 nó oculto usado, as precisões de classificação do treinamento da SCG e LM foram muito baixas.

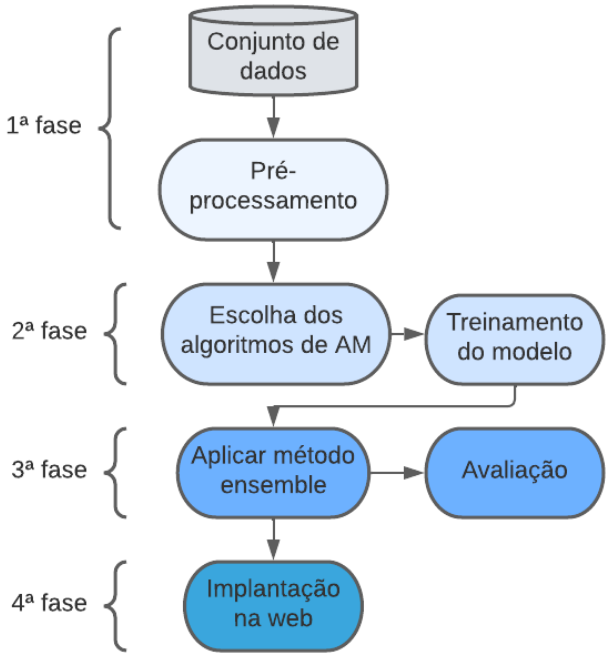
Peng *et al.* (2020) concluíram que quando o conjunto de dados não é pré-processado, a precisão obtida é de 98%. O aumento do número de nós ocultos melhora a precisão da classificação do modelo para a classe minoritária durante o treinamento, mas devido ao desequilíbrio do conjunto de dados, a precisão de classificação do Teste e Validação na classe minoritária foi extremamente baixa. No futuro, os autores pretendem melhorar o pré-processamento do conjunto de dados para resolver o problema de desbalanceamento.

O presente trabalho tem como semelhança a abordagem metodológica utilizada no trabalho de Emon *et al.* (2020), no qual diferentes métodos de AM serão aplicados para realizar a previsão, porém irá divergir com o uso de outro método de *ensemble* para avaliação, e na disponibilização do modelo em uma página *web*.

4 Metodologia

Neste capítulo será apresentada a metodologia adotada no presente trabalho para alcançar os objetivos propostos. Nesse sentido, emprega-se uma abordagem exploratória dividida em quatro fases, partindo da escolha de um *dataset* disponibilizado no Kaggle. As etapas estão descritas na Figura 5.

Figura 5 – Fluxograma da metodologia



Fonte: a autora

A primeira fase do trabalho consiste na escolha, preparação e pré-processamento dos dados. Essa etapa é de extrema importância para diminuir alguns problemas que são comumente encontrados em conjuntos de dados, tornando-os mais adequados para a utilização de certos algoritmos de AM.

Portanto, nesse sentido, durante a primeira fase foi realizada uma busca por dados incompletos. A ausência de valores para certos atributos é um problema comum em conjuntos de dados e podem estar ligados a diferentes causas. Os dados categóricos foram convertidos em dados numéricos para que dessa forma todo o conjunto de dados tenha valores números. Ainda na primeira fase, uma análise de correlação e de colinearidade foi realizada. Como o conjunto de dados é multivariado, ou seja, possuem mais de um atributos de entrada, essa análise tem como objetivo entender como diferentes atributos se relacionam e contribuem

para explicar as classes definidas na variável preditora. Realizou-se também balanceamento para a classe dependente.

Após realizado o pré-processamento, a segunda fase consiste na aplicação das técnicas de AM. Os dados são separados em treino e teste. Seis modelos de classificação foram escolhidos para o nível 0, conforme a técnica de *ensemble stacking*, e o treinamento realizado. Os modelos fizeram a predição da classe dependente (binária), referente ao atributo *'stroke'* e os resultados foram armazenados em um vetor contendo todas as saídas dos seis algoritmos que serviu com entrada para o sétimo algoritmo do próximo nível do *stacking*. Uma avaliação individual de cada algoritmo, utilizando o método de validação cruzada, também foi realizada.

Na terceira fase do trabalho, o método *ensemble* foi aplicado, que é uma técnica de AM que combina o resultado de múltiplos modelos em busca de produzir um melhor modelo preditivo, de forma a aumentar a acurácia do modelo com a junção de vários outros modelos. Em seguida, a estratégia de generalização em pilha (*stacking*), que consiste em combinar todos os algoritmos do nível 0 e fazer uma nova predição, foi aplicada e o algoritmo de nível 1 foi escolhido. Finalmente, com a predição final realizou-se a avaliação do modelo final.

A quarta fase consiste em disponibilizar o modelo por meio de uma página web, para que novas predições sejam feitas com novos dados de entrada.

5 Desenvolvimento

Neste capítulo será discorrido o passo a passo de cada etapa e as técnicas aplicadas no trabalho.

5.1 Visão geral do dataset

A plataforma do Kaggle disponibiliza diversas bases de dados, e entre elas o *Stroke Prediction Dataset* foi escolhido. O conjunto de dados apresenta dados de 5.110 pacientes com diferentes idades, e informações de relevância sobre a saúde do indivíduo. A Figura 6 apresenta cada um desses atributos que são as colunas da base de dados, a quantidade de informações e o tipo de dado.

Figura 6 – Informações do dataset

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   id                 5110 non-null   int64  
 1   gender             5110 non-null   object 
 2   age                5110 non-null   float64
 3   hypertension       5110 non-null   int64  
 4   heart_disease      5110 non-null   int64  
 5   ever_married       5110 non-null   object 
 6   work_type          5110 non-null   object 
 7   Residence_type     5110 non-null   object 
 8   avg_glucose_level  5110 non-null   float64
 9   bmi                4909 non-null   float64
 10  smoking_status     5110 non-null   object 
 11  stroke             5110 non-null   int64  
dtypes: float64(3), int64(4), object(5)
memory usage: 479.2+ KB
```

Fonte: a autora

Outra informação importante que precisa ser observada é o conteúdo em si do *dataset*, que tipo de informação está contida para identificar quais dados precisam ser transformados ou ajustados. A Figura 7 mostra as informações completas dos cinco primeiros exemplos do dataset.

Figura 7 – 5 primeiros exemplos do dataset

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	NaN	never smoked	1
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1

Fonte: a autora

A Tabela 2 a seguir contém o nome de cada atributo do *dataset* juntamente com a sua tradução para português.

Tabela 2 – Atributos e suas recpectivas traduções

Atributo	Tradução (PT-BR)
gender	Gênero
age	Idade
hypertension	Hipertensão
heart_disease	Doença do coração
ever_married	Já foi casado
work_type	Tipo de trabalho
residence_type	Tipo de residência
avg_glucose_level	Média do nível de glicose
bmi	IMC - Índice de Massa Corporal
smoking_status	Condição de fumante
stroke	Derrame (AVC)

Fonte: a autora

Com essas informações pode-se observar o seguinte sobre o conjunto de dados:

- Há dados faltantes para o atributo bmi conforme mostrado pelo item 9 da Figura 6;
- Atributos numéricos e categóricos estão presentes, sendo eles:
  - Categóricos: *gender, ever_married, work_type, residence_type, smoking_status*;
  - Numérico binário: *hypertension, heart_disease, stroke*;
  - Numérico contínuo: *age, avg_glucose_level, bmi*;
- A maioria dos dados são categóricos, o que indica que precisará de maior atenção para realizar as transformações.

5.2 Pré-processamento

5.2.1 Lidando com dados faltantes

Tendo em vista o tratamento dos dados, antes de realizar qualquer tipo de treinamento, foram realizadas várias etapas de pré-processamento com base nas observações realizadas na visualização geral do *dataset*. Identificou-se 201 dados faltantes para o atributo bmi. Para preencher esses dados ausentes, uma abordagem simples foi escolhida. Calculou-se a média dos valores conhecidos desse atributo, e assim, os dados faltantes foram preenchidos.

5.2.2 Transformando dados categóricos em dados numéricos

Para cada atributo do tipo categórico na base de dados, a transformação para tipo numérico foi realizada. Para exemplificar, o atributo *'ever_married'* possui como valores 'sim' e 'não', ao realizar a transformação, por meio de uma iteração, os valores possíveis passam a ser 0 e 1, respectivamente.

Esse processo foi realizado para todos os atributos categóricos e ao final, os dados do conjunto foram transformados em numéricos. A Figura 8 mostra como ficaram os cinco primeiros exemplos de dados após a transformação.

Figura 8 – 5 primeiros exemplos após transformação dos dados

	hypertension	heart_disease	avg_glucose_level	bmi	stroke	gender	ever_married	work_type	residence_type	smoking_status	age
0	0	1	228.69	36.6	1	0	0	0	0	0	7
1	0	0	202.21	28.7	1	1	0	1	1	1	6
2	0	1	105.92	32.5	1	0	0	0	1	1	8
3	0	0	171.23	34.4	1	1	0	0	0	2	5
4	1	0	174.12	24.0	1	1	0	1	1	1	8

Fonte: a autora

5.2.3 Correlação e colinearidade

Nessa etapa, aplicou-se os métodos da correlação e da colinearidade. Na biblioteca Pandas existe o método *corr()* que faz esse cálculo da correlação e na biblioteca do Python existe o método *variance_inflation_factor()* que detecta a multicolinearidade. Ambos os métodos foram aplicados ao *dataset*, após as devidas transformações para análise.

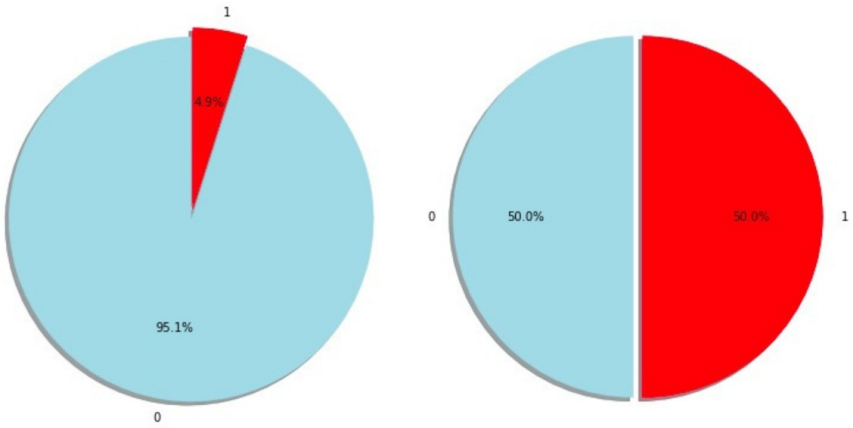
5.2.4 Balanceamento dos dados

Em todo o conjunto de dados há o total de 249 casos positivos para AVC (do inglês, *stroke*) e 4861 casos negativos, ou seja, apenas 5% de todos os dados são positivos. Esse desbalanceamento pode resultar na predição enviesada. Para balancear os dados, o método

de sobreamostragem (do inglês, *oversampling*) conhecido como *SMOTE* (técnica de sobreamostragem minoritária sintética) foi aplicado para gerar mais exemplos da classe minoritária. O método foi aplicado apenas nos dados de treinamento que foram separados previamente.

Existe a implementação da classe *SMOTE* na biblioteca do *scikit-learn*, cujo método *fit_resample* realiza o *oversampling* aumentando a quantidade de casos positivos para stroke. A Figura 9 mostra a proporção dos dados de AVC do *dataset* antes e depois do balanceamento com o método aplicado.

Figura 9 – Casos de AVC antes e depois do balanceamento com SMOTE

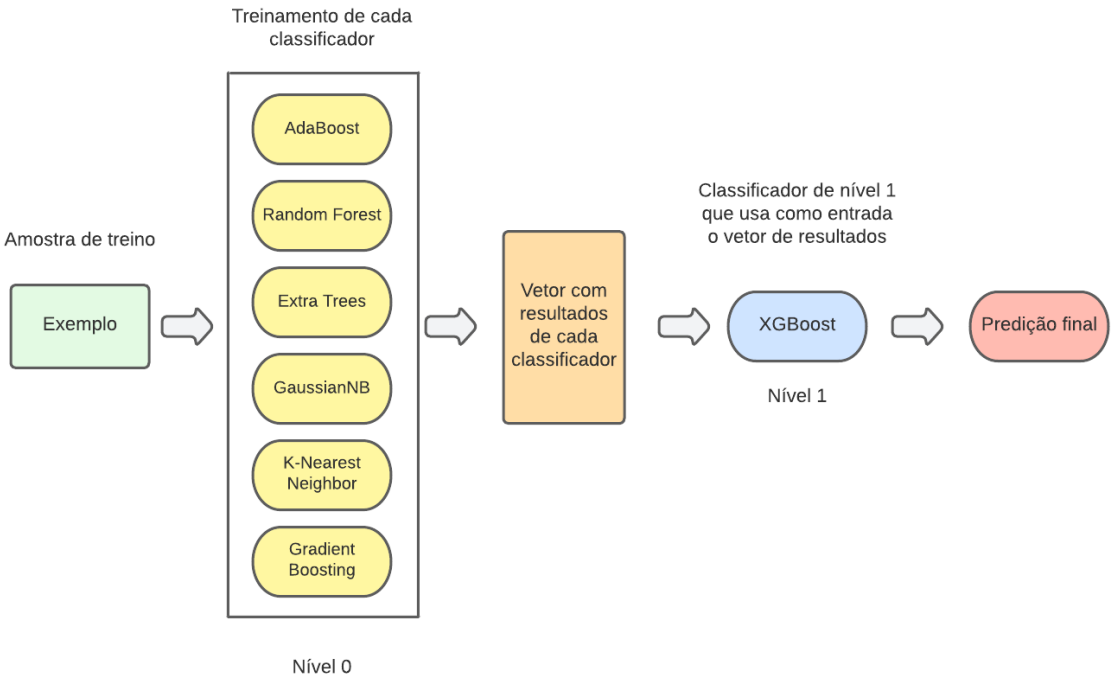


Fonte: a autora

5.3 Modelagem e treinamento

Para o treinamento dos dados pré-processados no presente trabalho, foram escolhidos seis algoritmos distintos de aprendizado de máquina para o nível 0 e um único algoritmo, diferente dos outros, para o nível 1, totalizando sete algoritmos. O objetivo disso é produzir resultados diferentes com porcentagem de acertos diferentes, para que um resultado melhor seja obtido, conforme prevê o método *ensemble* usando a técnica de *stacking*. Os algoritmos escolhidos para o primeiro nível foram: *AdaBoost, Random Forest, Extra Trees, Gaussian Naive Bayes, Gradient Boosting*, e *K-Nearest Neighbor*. Para o último nível, o algoritmo *XGBoost* foi escolhido. Foi utilizada a biblioteca *scikit-learn* do Python que possui todas as implementações para realizar a modelagem. A Figura 10 mostra como o *stacking* é implementado.

Figura 10 – Fluxo do método de *ensemble stacking*



Fonte: a autora

A base de dados foi dividida em 70% para treino e 30% para testes. Aplicando a técnica de *stacking*, no nível 0, todos os seis algoritmos recebem como entrada o mesmo conjunto de dados. Para todos os algoritmos foi definindo $k = 5$ para o *k-fold*, e o treinamento foi realizado com o *cross validation* por meio do método *cross_val_predict*, disponível na biblioteca do *scikit-learn* no módulo *sklearn.model_selection*. O método recebe como parâmetro um método estimador, o vetor de dados de entrada, o vetor de dados de saída para previsão, e número de *k-fold*. Ao final das predições, cada algoritmo teve seus resultados armazenados em um vetor. A validação da acurácia também foi realizada com o *cross-validation*, utilizando o método *cross_val_score* com valor de $k = 5$ para o *k-fold*. As porcentagens de acertos foram armazenadas em outros vetores.

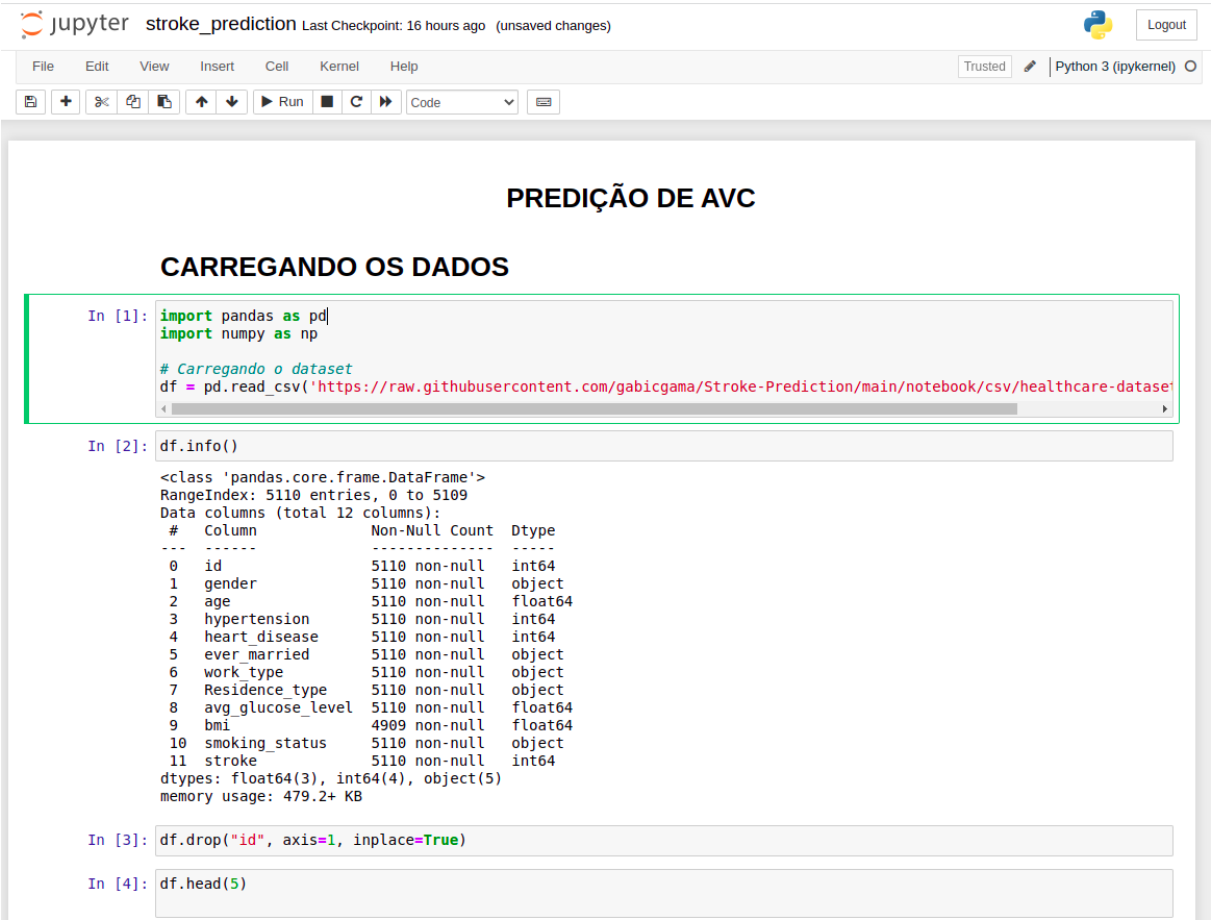
Na próxima etapa do *stacking*, todos os vetores de predições dos seis métodos de aprendizado, foram juntados em um único vetor para que o mesmo fosse usado como entrada no método *XGBoost*, que é o único classificador no nível 1. Utilizou-se o pacote *xgboost* do Python para modelagem utilizando a função *XGBClassifier* e treinamento com a função *fit*, que recebe como entrada o *dataset*, que no caso é o conjunto das previsões dos outros modelos.

5.4 Implantação do modelo em uma página da web

Nesta seção será discorrido sobre as ferramentas utilizadas para o desenvolvimento do modelo e implantação. O modelo foi desenvolvido por meio do Jupyter Notebook, escrito na linguagem Python. A API foi desenvolvida em Flask, um *framework* web do Python, e o *front-end* em Flasgger, que é uma extensão do Flask que permite criar páginas web.

O Jupyter Notebook é um aplicativo web de código aberto que oferece uma abordagem de *notebooks*, como se fossem cadernos, onde é possível escrever textos, códigos, visualização de gráficos, equações e entre outros, de forma simples e prática. Todos os processos descritos nas seções anteriores do Capítulo 5 foram feitas no Jupyter Notebook, conforme mostra a Figura 11.

Figura 11 – Desenvolvimento do modelo utilizando o Jupyter Notebook



Fonte: a autora

Para exportação do modelo treinado, foi utilizado o módulo *pickle* do Python. Ele realiza a serialização binária dos dados para que seja possível enviar o modelo pela rede. Um arquivo chamado "*model.pkl*" foi criado e utilizado neste arquivo a função *pickle.load* dentro da Flask API.

Flask é um *framework* destinado principalmente a pequenas aplicações com requisitos mais simples, como por exemplo, a criação de um site básico. API significa *Application Programming Interface* (Interface de Programação de Aplicação), que é uma interface desenvolvida para se comunicar com outras aplicações.

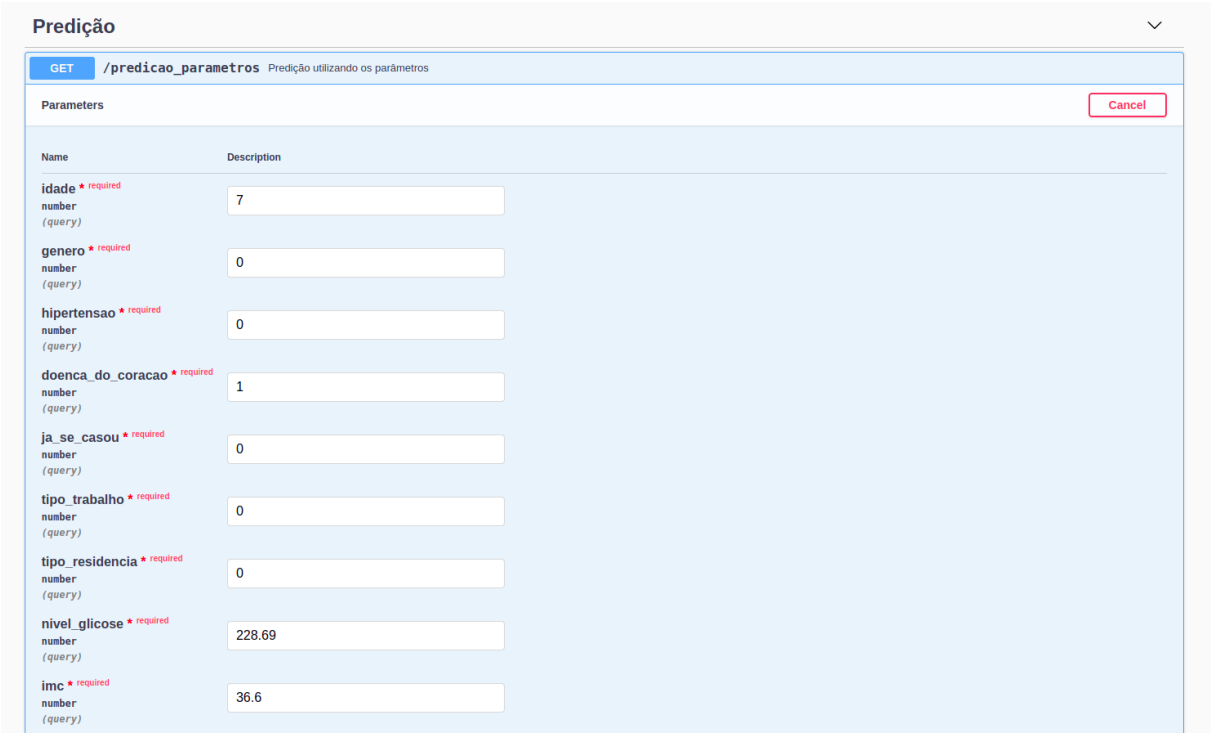
Por fim, para visualizar a API criada em uma interface gráfica, foi utilizado o Flasgger, uma extensão do Flask que permite a criação de interfaces web sem a necessidade de dominar o desenvolvimento web.

Para criar a interface, o Flasgger define uma sintaxe que devemos seguir. Existem diversos exemplos e *templates* que podem ser encontrados na internet. Para este trabalho utilizou-se um *template* simples. A página criada é mostrada nas figuras 12 a 14 a seguir:

Figura 12 – Visualização da interface da API por meio de uma página web

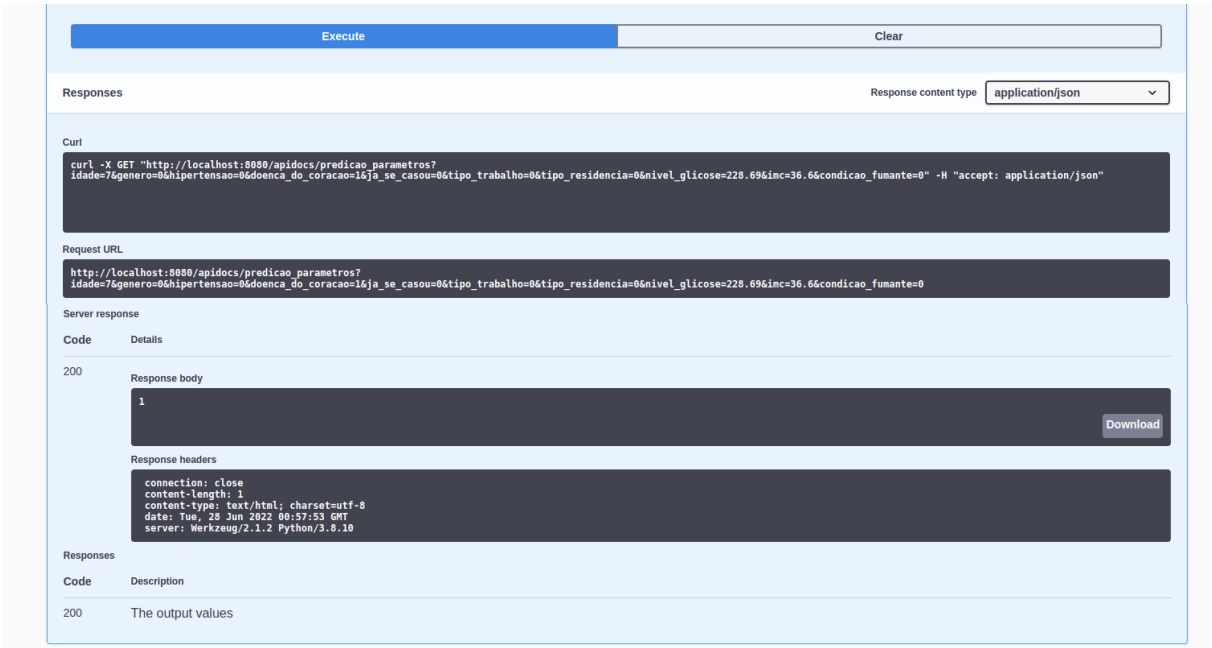
Fonte: a autora

Figura 13 – Visualização da interface da API - Entrada dos parâmetros



Fonte: a autora

Figura 14 – Visualização da interface da API - Resposta da API



Fonte: a autora

Todo o processo de implantação do modelo com Flask e Flasgger foi feito baseado em um artigo do Domingues (2020). O *script* do *notebook* pode ser visualizado no Apêndice A e

código fonte da API no Apêndice B.

6 Resultados

Neste capítulo será discorrido sobre a análise e visualização do conjunto de dados trabalhados ao longo do projeto, bem como análises dos resultados encontrados referentes aos testes realizados durante o desenvolvimento. São apresentados gráficos para ilustrar a base de dados, tabelas contendo informações sobre a correlação entre os atributos dos conjuntos. Ademais apresentam-se também as métricas de avaliação dos algoritmos de AM, como precisão, revocação e acurácia de cada um dos classificadores individualmente e após combinados com a técnica *ensemble* de generalização em pilha (*stacking*).

6.1 Visualização dos dados

A visualização dos dados é importante para entender um pouco sobre o conjunto de dados que será utilizado no trabalho. Dessa forma, algumas conclusões e inferências podem ser feitas. Prosseguindo com a visualização, as Tabela 3 até a Tabela 9 apresentam informações pertinentes à quantidade de pacientes de determinado gênero, situação do casamento, se possuem hipertensão ou não, tipo de trabalho, tipo de residência, e se é fumante, respectivamente.

Tabela 3 – Gênero

Feminino	Masculino	Outro
2994	2115	1

Tabela 4 – Hipertensão

Possui hipertensão	Não possui hipertensão
498	4612

Tabela 5 – Situação do casamento

Já se casou	Nunca casou
3353	1757

Tabela 6 – Tipo de trabalho

Setor Privado	Autônomo	Setor infantil	Setor público	Nunca trabalhou
2925	819	687	657	22

Tabela 7 – Tipo de residência

Urbana	Rural
2596	2514

Tabela 8 – Condição de fumante

Nunca fumou	É fumante	Parou de fumar	Não soube dizer
1892	789	885	1544

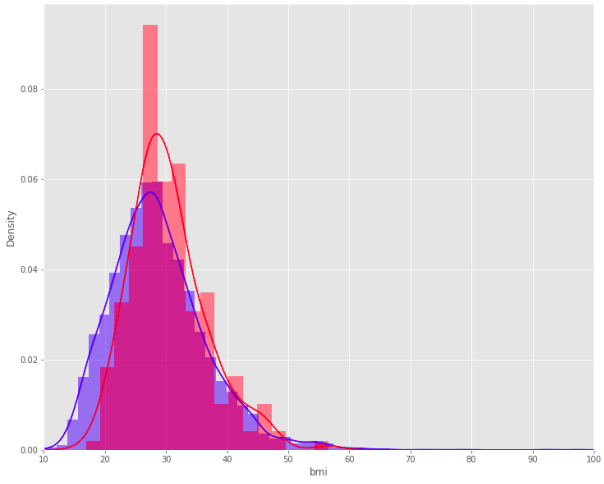
Tabela 9 – Doenças do coração

Já ocorreu	Nunca ocorreu
276	4834

Conforme as tabelas apresentadas, pode-se observar que o conjunto de dados possui uma quantidade maior de mulheres, com 2.994 pessoas do gênero feminino e 2.115 do gênero masculino. Há menos casos de hipertensos, 498 pacientes informaram terem hipertensão enquanto que 4.612 não possuem. Pode-se ver que a razão entre pessoas casadas e não casadas é de 2:1. A maioria das pessoas trabalham no setor privado. O tipo de residência está bem equivalente. E muitas pessoas informaram nunca terem fumado, porém também houveram muitas pessoas que não souberam dizer o seu nível de fumante.

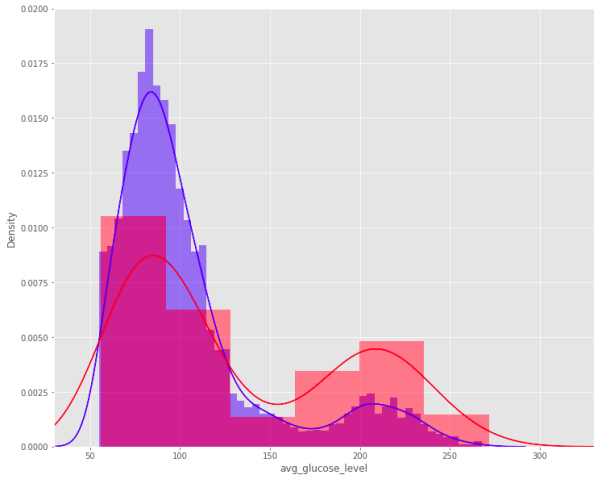
A seguir temos os gráficos de distribuição que mostram a relação entre os casos de AVC com o índice de massa corporal (IMC), na Figura 15, com o nível de glicose no corpo, na Figura 16, e com as idades dos pacientes, na Figura 17. Em todos esses gráficos a cor azul representa os casos negativos de AVC, enquanto que o vermelho representa os positivos.

Figura 15 – Casos sem AVC vs com AVC por IMC



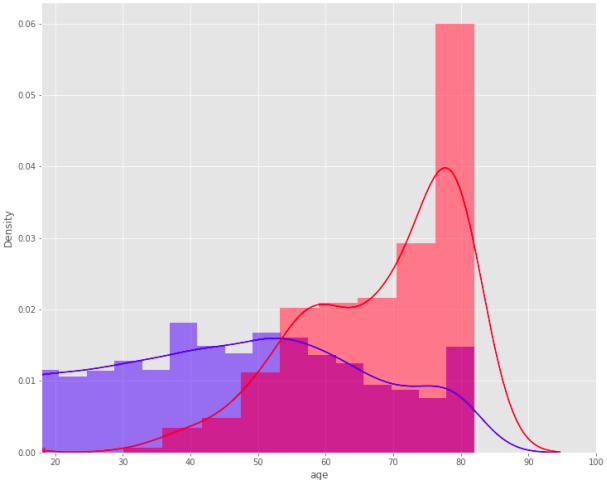
Fonte: a autora

Figura 16 – Casos sem AVC vs com AVC por nível de glicose



Fonte: a autora

Figura 17 – Casos sem AVC vs com AVC por idade



Fonte: a autora

A partir do gráfico da Figura 15 podemos deduzir que a densidade de pessoas acima do peso, isto é, com IMC acima de 30 (em inglês, body mass index - BMI), que sofreram de AVC é maior. Na Figura 16 é mostrado que a maioria das pessoas com o nível de glicemia inferior a 100 não sofreram de AVC, enquanto que acima de 150, a maioria das pessoas sofreu. Por fim, a partir da Figura 17 pode-se inferir que pacientes abaixo de 50 anos, em sua maioria, nunca sofreram de AVC, enquanto que a maioria das pessoas com idade superior a 50 anos já.

Algumas deduções a partir da visualização dos dados foram feitas:

- baseado nos dados, o gênero não afeta a probabilidade de se ter AVC;

- há uma grande correlação entre idade e risco de AVC. Quase todos os médicos dizem que essa doença está ligada a idade, entretanto ela também pode ocorrer em crianças, em alguns casos raros;
- a presença de hipertensão interfere no risco de AVC;
- pessoas casadas possuem mais chance de ter AVC, o que pode ser uma coincidência;
- viver em zona rural ou urbana não interfere no risco;
- fumar também não afeta o risco de AVC, entretanto pode causar outros tipos de doenças;
- obesidade, além de causar diversas doenças, influencia no risco de ocorrer AVC;
- a diabetes, causada pelo nível de glicemia, tem certo impacto no risco de AVC.

6.2 Balanceamento

Inferiu-se, conforme foi apresentado na Figura 9, que a distribuição dos casos positivos de AVC em relação aos negativos está muito desbalanceada, com apenas 5% de casos positivos. Qualquer modelo empregado para realizar uma previsão aleatória implicaria em uma precisão bastante enviesada. Portanto, ao modelar e treinar dados, é necessário fazer sobreamostragem ou subamostragem para obter os melhores resultados. Ao aplicar o método de *SMOTE*, da biblioteca *scikit-learn*, que é uma técnica de sobreamostragem, obteve o balanceamento dos dados. A Tabela 10 mostra a quantidade de amostras com AVC antes de depois da aplicação da técnica.

Tabela 10 – Antes e depois do balanceamento dos dados com SMOTE

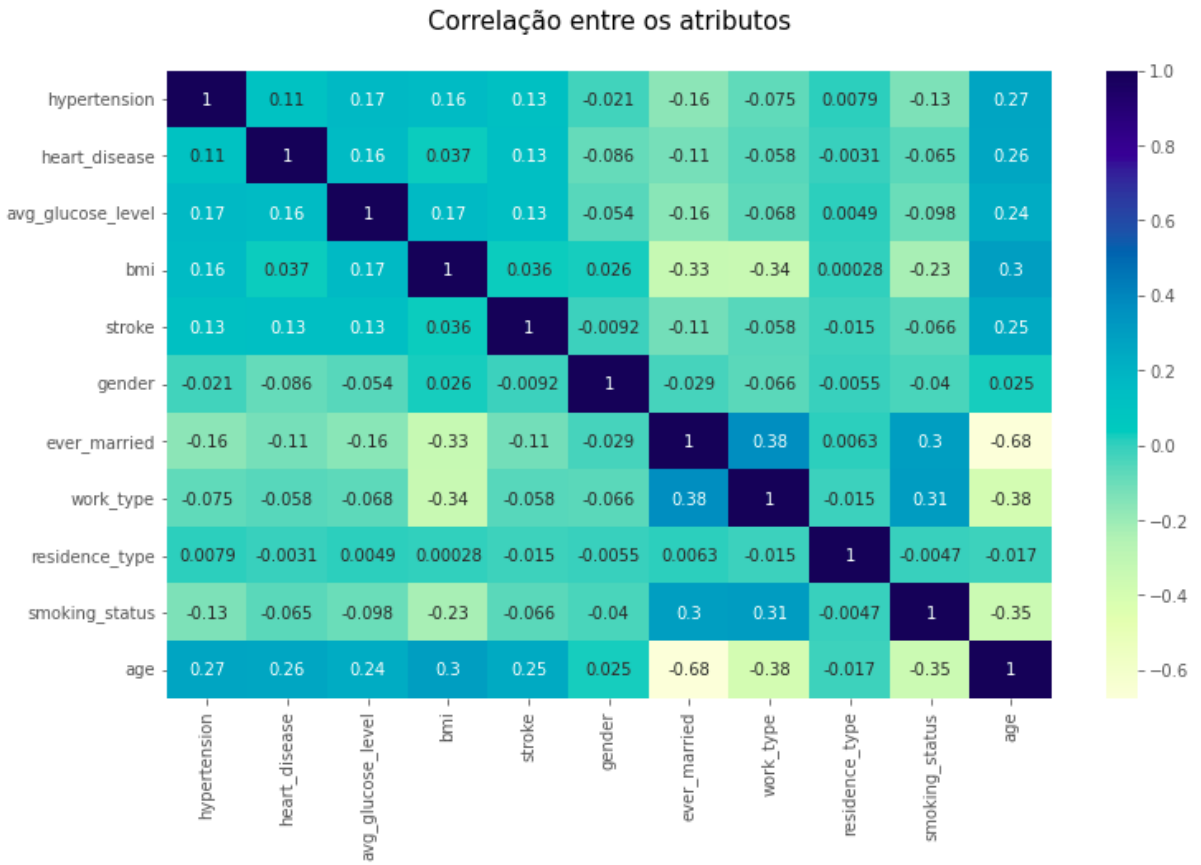
	Já ocorreu AVC	Nunca ocorreu AVC	Total
Sem SMOTE	249	4861	5110
Com SMOTE	3408	3408	6816

Fonte: a autora

6.3 Matriz de Correlação

A Figura 18, apresenta a matriz de correlação de Pearson entre todos os atributos. Essa matriz mede o grau de relação linear entre cada par de variáveis. Os valores de correlação podem variar entre -1 e 1. Se duas variáveis tendem a aumentar ou diminuir juntas, o valor da correlação é positivo. Se uma diminui enquanto a outra aumenta, e vice-versa, o valor é negativo.

Figura 18 – Matriz de correlação



Fonte: a autora

A matriz é usada para medir a força e a direção da relação entre os pares de variáveis. Um valor de correlação alto indica que elas medem a mesma característica. Se as variáveis não estão correlacionadas, isso pode indicar que elas medem diferentes características ou que não estão muito bem definidas.

Conforme a Figura 18, todos os valores estão muito baixos, indicando uma relação fraca entre os atributos, sendo o valor mais alto de -0,68 entre os atributos *'age'* e *'ever_married'*, podendo ser considerado como moderado. Portanto, nenhuma característica explica mais a ocorrência de AVC do que a outra.

6.4 Multicolinearidade - VIF

A Figura 19, mostra o fator de inflação da variância (VIF), que é uma maneira de medir a multicolinearidade entre dois ou mais atributos. Uma regra prática e presente na literatura, sugere que um VIF $>$ 10 indica multicolinearidade. Para o presente conjunto de dados, apenas o BMI, ou IMC, apresentou valor maior que 10, então essa variável não será excluída.

Figura 19 – Fator de Inflação da Variância

	feature	VIF
0	age	8.191334
1	gender	2.356298
2	hypertension	1.214736
3	heart_disease	1.166515
4	ever_married	2.340551
5	work_type	1.912006
6	residence_type	1.918026
7	avg_glucose_level	6.749972
8	bmi	10.822353
9	smoking_status	3.306998
10	stroke	1.137361

Fonte: a autora

6.5 Resultados da previsão de cada modelo de AM

Nesta seção serão discutidos os resultados dos treinos e das previsões referentes a cada um dos algoritmos classificadores escolhidos, e o resultado após a aplicação do método *ensemble* de *stacking*. Os métodos de classificação do primeiro nível são os seguintes: AB, RF, ET, GNB, KNN, GBC e para o segundo nível, o algoritmo escolhido foi *XGBoost*.

A seguir, serão apresentadas tabelas 11 até a 16 com as métricas de avaliação de cada um dos classificadores, obtidas pelo uso do método *cross_validate* do pacote *sklearn.model_selection*. O *'fit_time'* e *'score_time'* representam, respectivamente, o tempo gasto para treino e para avaliação. A métrica *'test_precision_macro'* descreve o valor da precisão dos testes, que indica dentre todas as classificações de classe Positivo que o modelo fez, quantas estão corretas. Já a métrica *'test_recall_macro'* descreve o valor do *recall* ou revocação que indica dentre todas as situações de classe Positivo como valor esperado, quantas estão corretas.

Tabela 11 – Validação cruzada do AdaBoost

	fit_time	score_time	test_precision_macro	test_recall_macro
0	1.90057135	0.1347487	0.86201938	0.85991555
1	1.86626101	0.14132476	0.85600234	0.8546226
2	1.85100269	0.13819575	0.84315787	0.84088364
3	1.8400631	0.13893938	0.85425721	0.85276214
4	1.83273005	0.136801	0.8464573	0.84585605
média:	1.85812563	0.13800191	0.85237882	0.85080799

Fonte: a autora

A Tabela 11 mostra que para o classificador *AdaBoost* as médias obtidas para o tempo

de treino foram de 1,85 segundos, tempo de avaliação de 0,13 segundos, uma precisão de 85% e *recall* de 85%.

Tabela 12 – Validação cruzada do Random Forest

	fit_time	score_time	test_precision_macro	test_recall_macro
0	1.37230182	0.08072114	0.82047973	0.81072319
1	1.40020084	0.08992243	0.86206445	0.85148869
2	1.34467363	0.08728909	0.83421829	0.82120188
3	1.36929536	0.08486391	0.84461709	0.82760076
4	1.45321727	0.15216541	0.84625686	0.83682574
média:	1.38793778	0.09899239	0.84152728	0.82956805

Fonte: a autora

A Tabela 12 mostra que para o classificador *Random Forest* as médias obtidas para o tempo de treino foram de 1,38 segundos, tempo de avaliação de 0,09 segundos, uma precisão de 84% e *recall* de 83%.

Tabela 13 – Validação cruzada do Extra Trees

	fit_time	score_time	test_precision_macro	test_recall_macro
0	2.03866649	0.30985403	0.80382104	0.79533263
1	1.32920289	0.20635438	0.83062322	0.82440422
2	0.91999364	0.20615196	0.82408487	0.81602252
3	0.94956279	0.20876455	0.8321284	0.82158576
4	0.91820383	0.20630336	0.83084218	0.8236517
média:	1.23112592	0.22748565	0.82429994	0.81619936

Fonte: a autora

A Tabela 13 mostra que para o classificador *Extra Trees* as médias obtidas para o tempo de treino foram de 1,23 segundos, tempo de avaliação de 0,22 segundos, uma precisão de 82% e *recall* de 81%.

Tabela 14 – Validação cruzada do Gaussian Naive Bayes

	fit_time	score_time	test_precision_macro	test_recall_macro
0	0.00841355	0.00478554	0.81844293	0.78127506
1	0.00397706	0.00462508	0.83162868	0.79691604
2	0.00388718	0.00389338	0.81978817	0.78539531
3	0.00372148	0.0038836	0.81702039	0.77885415
4	0.00380254	0.00387907	0.82583815	0.79326107
média:	0.00476036	0.00421333	0.82254366	0.78714032

Fonte: a autora

A Tabela 14 mostra que para o classificador *Gaussian Naive Bayes* as médias obtidas para o tempo de treino foram de 0,004 segundos, tempo de avaliação de 0,004 segundos, uma precisão de 82% e *recall* de 78%.

Tabela 15 – Validação cruzada do k-Nearest Neighbor

	fit_time	score_time	test_precision_macro	test_recall_macro
0	0.01568818	0.05127144	0.89782355	0.87592852
1	0.00909901	0.05089402	0.89362238	0.87339698
2	0.00895262	0.05061054	0.89496631	0.87333348
3	0.00892258	0.05064178	0.8935177	0.87382919
4	0.00913811	0.04939151	0.89202175	0.87341313
média:	0.01036009	0.05056185	0.89439033	0.87398025

Fonte: a autora

A Tabela 15 mostra que para o classificador *k-Nearest Neighbor* as médias obtidas para o tempo de treino foram de 0,01 segundos, tempo de avaliação de 0,05 segundos, uma precisão de 89% e *recall* de 87%.

Tabela 16 – Validação cruzada do Gradient Boosting

	fit_time	score_time	test_precision_macro	test_recall_macro
0	4.93636322	0.0213089	0.93785516	0.93468932
1	4.89721799	0.0203588	0.94437156	0.94342816
2	4.87885594	0.0211556	0.94309781	0.94137671
3	4.85571146	0.02215743	0.9472463	0.94658952
4	4.82659245	0.0204258	0.93934752	0.93906372
média:	4.87894821	0.02108130	0.94238367	0.94102948

Fonte: a autora

A Tabela 16 mostra que para o classificador *Gradient Boosting* as médias obtidas para o tempo de treino foram de 4,87 segundos, tempo de avaliação de 0,02 segundos, uma precisão de 94% e *recall* de 94%.

A seguir, a Figura 20 coloca em comparação o valor da acurácia da validação cruzada com $k = 5$ para *k-fold* de todos os classificadores, valores esses obtidos utilizando o método *cross_val_score* do *scikit-learn*. Realizando a média da validação cruzada dos algoritmos, conforme Tabela 17, observou-se que o classificador *Gradient Boosting* obteve o maior resultado com acurácia de 93,97%, em seguida o classificador KNN, com 87,39%. Quem obteve a menor acurácia foi o classificador GNB, com 78,71%.

Figura 20 – Acurácia de todos os classificadores

	AdaBoost	RandomForest	ExtraTrees	GaussianNB	K-NearestNeighbor	GradientBoosting
0	0.859971	0.815249	0.790323	0.781525	0.876100	0.934018
1	0.854732	0.853265	0.823184	0.797506	0.873808	0.937638
2	0.840792	0.822450	0.815847	0.785033	0.873074	0.941306
3	0.852531	0.822450	0.826853	0.777696	0.873074	0.941306
4	0.845928	0.836390	0.820983	0.793837	0.873808	0.944241

Fonte: a autora

Tabela 17 – Média dos resultados da validação cruzada dos algoritmos

	AB	RF	ET	GNB	KNN	GBC
média:	0.8507908	0.8292215	0.8154434	0.7871194	0.8739728	0.9397018

Fonte: a autora

6.6 Resultados da previsão da combinação dos modelos com a técnica de ensemble stacking

Conforme explicado no Capítulo 5, a técnica de *stacking* consiste em pegar o vetor de resultados de todos os algoritmos classificadores do nível 0 e fazer uma nova predição com o algoritmo do próximo nível, sendo este o *XGBoost*. Após realizado o treinamento desse algoritmo, a validação cruzada também foi aplicada no mesmo para fins de análise do desempenho. A Figura 21 contém o resultado da acurácia do método *ensemble stacking*, que faz a previsão a partir da previsão de todos os classificadores, em comparação com a acurácia individual de cada classificador. Pode-se perceber que a acurácia do *stacking*, conforme a média de validação cruzada presente na tabela, foi superior a de todos os demais algoritmos, com o resultado de 94,45%.

Figura 21 – Acurácia de todos os classificadores e do modelo de stacking

	AdaBoost	RandomForest	ExtraTrees	GaussianNB	K-NearestNeighbor	GradientBoosting	StackingModel
0	0.859971	0.815249	0.790323	0.781525	0.876100	0.934018	0.943548
1	0.854732	0.853265	0.823184	0.797506	0.873808	0.937638	0.944974
2	0.840792	0.822450	0.815847	0.785033	0.873074	0.941306	0.947909
3	0.852531	0.822450	0.826853	0.777696	0.873074	0.941306	0.946442
4	0.845928	0.836390	0.820983	0.793837	0.873808	0.944241	0.939839

Fonte: a autora

Tabela 18 – Média dos resultados da validação cruzada do modelo stacking

	StackingModel
média:	0.9445424

Fonte: a autora

7 Conclusão

Neste trabalho, realizou-se a predição da ocorrência ou não de Acidente Vascular Cerebral (AVC) em pacientes, utilizando uma base de dados disponível no Kaggle, a partir do desenvolvimento de um modelo de aprendizado de máquina, aplicando as técnicas que foram apresentadas na metodologia do presente trabalho, podendo auxiliar na detecção de prováveis pacientes terem um AVC no futuro.

Para atingir o objetivo proposto, no primeiro momento buscou-se fazer uma análise exploratória dos dados com o intuito de identificar e reduzir problemas que são comuns ao se desenvolver um modelo de predição. Verificou-se que o conjunto de dados havia informações faltantes, dados categóricos precisam ser convertidos para numéricos e que havia um alto desbalanceamento entre os dados referentes a pacientes que já tiveram ou não AVC. Aplicando as técnicas de pré-processamento, obteve-se um conjunto de dados tratado, completo, com todos os dados números e com o atributo alvo de predição balanceado, pronto e adequado para realizar o treinamento dos modelos escolhidos. A Matriz de Correlação de Pearson apresentou valores muito baixos, indicando uma relação fraca entre os atributos, portanto nenhuma característica explica mais a ocorrência de AVC do que a outra.

Conforme a metodologia proposta, os algoritmos AB, RF, EX, GNB, KNN e GBC foram treinados, todos utilizando as mesmas entradas de treino e teste, e a avaliação de desempenho foi realizada, levando em consideração métricas importantes como precisão, acurácia e *recall*. Com a aplicação da técnica de validação cruzada em cada um dos algoritmos, foi possível analisar e concluir que o classificador *Gradient Boosting* obteve o maior resultado com acurácia de 93,97%, em seguida o classificador *K-Nearest Neighbor*, com 87,39%. O método que obteve a menor acurácia foi o classificador *Gaussian Naive Bayes*, com 78,71%.

Seguindo com a metodologia, o próximo passo empregado foi a aplicação de um método de *ensemble*, para combinar as predições de todos os seis algoritmos e realizar uma predição final. A técnica de generalização em pilha (*stacking*) aplicada, utilizando o algoritmo *XGBoost* para fazer uma predição a partir da combinação das predições anteriores, apresentou um desempenho melhor, com uma acurácia de 94,45%. Dessa forma, pode-se concluir que a aplicabilidade do método *ensemble* se justificava como um procedimento benéfico para obter melhores resultados, visto que, em comparação com os resultados individuais dos algoritmos, o resultado a partir da combinação das predições foi superior.

Com isso, a utilização de métodos de aprendizado de máquina, com a metodologia proposta neste trabalho, como uma ferramenta para a predição de dados de AVC, mostrou-se uma opção promissora apresentando bons resultados de desempenho. Novas predições com novos dados de pacientes podem ser realizadas por meio da interface da API desenvolvida e

disponibilizada em uma página web.

Como trabalho futuro, as análises e técnicas aplicadas neste trabalho podem ser adotadas para uma outra base de dados coletadas do sistema de saúde brasileiro, podendo ainda realizar outras escolhas e combinações de algoritmos diferentes para a aplicação do método de *ensemble stacking*, para resolver problemas de classificação ou de regressão.

Referências

ABRAMCZUK, B.; VILLELA, E. A luta contra o avc no brasilo. In: _____. [s.n.], 2009. Disponível em: <https://www.paho.org/bra/index-.php?option=com_content&view=article&id=5638:10-principais-causas-de-morte-no-mundo&Itemid=0>. Acesso em: 10 de fev. de 2021. Citado na página 1.

AHA, D. W.; KIBLER, D.; ALBERT, M. K. Instance-based learning algorithms: Machine learning 6. p. 37–66, 1991. Citado na página 9.

ALI, K.; PAZZANI, M. Error reduction through learning multiple descriptions. **Machine Learning**, v. 24, 11 1997. Citado na página 11.

BARNARD, G. A. New methods of quality control. Journal Royal Statictical Society - Series A, p. 126–255, 1963. Citado na página 11.

BOND, L. Lembrado hoje, dia mundial do avc serve de alerta à população. In: _____. [s.n.], 2020. Disponível em: <https://agenciabrasil.ebc.com.br/saude/noticia/2020-10/lembrado-hoje-dia-mundial-do-avc-serve-de-alerta-populacao>. Acesso em: 30 de jan. de 2021. Citado na página 1.

BREIMAN, L. Stacked regressions: Machine learning. p. 24:49–64, 1997. Citado na página 14.

BREIMAN, L. Random forests: Machine learning. p. 45:5–32, 2001. Citado na página 7.

CHEN, T.; GUESTRIN, C. Xgboost: A scalable tree boosting system. In: . New York, NY, USA: Association for Computing Machinery, 2016. (KDD '16). ISBN 9781450342322. Disponível em: <https://doi.org/10.1145/2939672.2939785>. Citado na página 10.

DOMINGUES, E. Da modelagem ao deploy — modelos. In: _____. [s.n.], 2020. Disponível em: <https://eduardo-p-domingues.medium.com/da-modelagem-ao-deploy-modelos-691e87422007>. Acesso em: 30 maio. 2022. Citado na página 28.

Emon, M. U. *et al.* Performance analysis of machine learning approaches in stroke prediction. In: **2020 4th International Conference on Electronics, Communication and Aerospace Technology (ICECA)**. [S.l.: s.n.], 2020. p. 1464–1469. Citado 2 vezes nas páginas 17 e 18.

ESTEVES, R. S.; LORENA, A. C.; NASCIMENTO, M. Z. **Aplicação de técnicas Aprendizado de Máquina na Classificação de Imagens Mamográficas**. Simpósio de Iniciação Científica da Universidade Federal do ABC, 2009. Disponível em: <http://ic.ufabc.edu.br-/II_SIC_UFABC/resumos/paper_5_150.pdf>. Citado na página 4.

FACELI, K. *et al.* **Inteligência Artificial: Uma Abordagem de Aprendizado de Máquina**. Rio de Janeiro: LTC, 2011. Citado 5 vezes nas páginas 4, 5, 7, 11 e 13.

Freund, Y.; Schapire, R. Experiments with a new boosting algorithm. In: **Proc. of the 13th International Conference**. [S.l.: s.n.], 1996. p. 148–156. Citado na página 6.

FRIEDMAN, J. H. **Greedy function approximation: a gradient boosting machine**. [S.l.]: Annals of statistics, 2001. 1189-1232 p. Citado 2 vezes nas páginas 9 e 10.

GEURTS, P.; ERNST, D.; WEHENKEL, L. Extremely randomized trees: Machine learning. v. 63, p. 3–42, 2006. Disponível em: <https://doi.org/10.1007/s10994-006-6226-1>. Citado na página 8.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>. Citado na página 5.

Jeena, R. S.; Kumar, S. Stroke prediction using svm. In: **2016 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)**. [S.l.: s.n.], 2016. p. 600–602. Citado 2 vezes nas páginas 16 e 17.

Kansadub, T. *et al.* Stroke risk prediction model based on demographic data. In: **2015 8th Biomedical Engineering International Conference (BMEiCON)**. [S.l.: s.n.], 2015. p. 1–3. Citado na página 16.

LEWIS, D. D. Naive (bayes) at forty: The independence assumption in information retrieval. In: NÉDELLEC, C.; ROUVEIROL, C. (Ed.). **Machine Learning: ECML-98**. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998. p. 4–15. Citado na página 8.

MACHADO, W. dos S. **Avaliação de modelos de classificação automática de atividades diárias para dispositivos de baixo custo**. 68 p. Monografia (Mestrado em Modelagem Computacional do Conhecimento) — Universidade Federal de Alagoas, Maceió, 2020. Citado na página 8.

MICHIE, D. *et al.* Machine learning, neural and statistical classification. In: . [S.l.: s.n.], 1994. p. 600–602. Citado na página 11.

OLIVEIRA, A. R. **Comparação de algoritmos de aprendizagem de máquina para construção de modelos preditivos de diabetes não diagnosticado**. Monografia (Programa de Pós-Graduação em Computação) — Universidade Federal do Rio Grande do Sul, Rio Grande do Sul, 2016. Citado na página 1.

OPAS. 10 principais causas de morte no mundo. In: _____. [s.n.], 2018. Disponível em: <https://www.paho.org/bra/index.php?option=com_content&view=article&id=5638:10-principais-causas-de-morte-no-mundo&Itemid=0>. Acesso em: 30 jan. 2021. Citado na página 1.

PEDREGOSA, F. *et al.* Scikit-learn: Machine learning in Python. **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011. Citado na página 15.

Peng, C. C. *et al.* Artificial neural network application to the stroke prediction. In: **2020 IEEE 2nd Eurasia Conference on Biomedical Engineering, Healthcare and Sustainability (ECBIOS)**. [S.l.: s.n.], 2020. p. 130–133. Citado 2 vezes nas páginas 17 e 18.

Singh, M. S.; Choudhary, P. Stroke prediction using artificial intelligence. In: **2017 8th Annual Industrial Automation and Electromechanical Engineering Conference (IEMECON)**. [S.l.: s.n.], 2017. p. 158–161. Citado na página 17.

TAKAKURA, A. M. *et al.* Use of machine learning in the medical diagnosis of pathologies. In: **Colloquium Exactarum**. [S.l.: s.n.], 2018. v. 10, n. 1, p. 78–90. Citado na página 2.

WOLPERT, D. Stacked generalization: Neural networks. p. 5:241–260, 1992. Citado na página 12.

WOLPERT, D.; MACREADY, W. G. No free lunch theorems for optimization. IEEE Trans. Evolutionary Computation, p. 67–82, 1997. Citado na página 11.

Apêndices

APÊNDICE A – Script do notebook

```
# PREVISÃO DE AVC

# CARREGANDO OS DADOS


import pandas as pd
import numpy as np

# Carregando o dataset
df = pd.read_csv('https://raw.githubusercontent.com/gabicgama/Stroke-Predictio
                         /main/notebook/csv/healthcare-dataset-stroke-data.csv'

df.info()

df.drop("id", axis=1, inplace=True)

df.head(5)

df.describe()

#Como BMI tem valores missing vamos preencher com a moda pegando a
#ocorrencia zero pois é  primeira do array
#from scipy.stats import mode

#Preenchendo valores de BMI com a média
df['bmi'] = df['bmi'].fillna(round (df['bmi'].median(), 2))
df.isnull().sum()

df.info()

df.head(5)

df.describe()

"""# PREPARAÇÃO E TRATAMENTO"""
```

```
df.head()

# Dicionário para o genero - Transformando variável categórica
dict_gender = {}

i = 0

# Loop pelos dados dos gêneros
for genero in df['gender'].values:
    if genero not in dict_gender:
            dict_gender[genero] = i
            i += 1

print(dict_gender)

# Dicionário para o casado - Transformando variável categórica
dict_ever_married = {}

i = 0

# Loop pelos dados dos casados
for casado in df['ever_married'].values:
    if casado not in dict_ever_married:
            dict_ever_married[casado] = i
            i += 1

print(dict_ever_married)

# Dicionário para o tipo de trabalho
dict_work_type = {}

i = 0

# Loop pelos dados dos gêneros
for tipo in df['work_type'].values:
    if tipo not in dict_work_type:
            dict_work_type[tipo] = i
            i += 1
```

```
print(dict_work_type)

# Dicionário para o tipo de residência
dict_residence_type = {}

i = 0

# Loop pelos dados dos tipos de residência
for tipo in df['Residence_type'].values:
    if tipo not in dict_residence_type:
              dict_residence_type[tipo] = i
              i += 1

print(dict_residence_type)

# Dicionário para o smoke status
dict_smoking_status = {}

i = 0

# Loop pelos dados dos status do fumante
for status in df['smoking_status'].values:
    if status not in dict_smoking_status:
              dict_smoking_status[status] = i
              i += 1

print(dict_smoking_status)

#Define uma função genérica para transformar variável categoriacas
#em numéricas usando a função replace
def converte(col, codeDict):
  colCoded = pd.Series(col, copy=True)
  for key, value in codeDict.items():
    colCoded.replace(key, value, inplace=True)
  return colCoded

#convertendo as variáveis categoricas
df["genero"] = converte(df["gender"], dict_gender)
```

```
df["casado"] = converte(df["ever_married"], dict_ever_married)
df["tipo_trabalho"] = converte(df["work_type"], dict_work_type)
df["tipo_residencia"] = converte(df["Residence_type"], dict_residence_type)
df["fumante"] = converte(df["smoking_status"], dict_smoking_status)

#Criando faixas de idades
lst = []
for age in df['age'].values:
    if age < 18.0:
        lst.append(1)
    elif age < 25.0:
        lst.append(2)
    elif age < 35.0:
        lst.append(3)
    elif age < 45.0:
        lst.append(4)
    elif age < 55.0:
        lst.append(5)
    elif age < 65.0:
        lst.append(6)
    elif age < 75.0:
        lst.append(7)
    else:
        lst.append(8)

(df['idade']) = lst

df['idade'].count

#Apagando as colunas categoricas
del df['age']
del df['gender']
del df['ever_married']
del df['work_type']
del df['Residence_type']
del df['smoking_status']

#printando nomes das colunas
#for atb in df:
```

```
#    print(atb)

import statsmodels.formula.api as smf
# Criando o Modelo de Regressão usando o algoritmo OLS
estimativa = smf.ols(formula = 'stroke ~ bmi', data = df)

# Treinando o Modelo de Regressão
modelo = estimativa.fit()

# Imprimindo o resumo do modelo
print(modelo.summary())

df.rename(columns={'idade':'age','genero':'gender','casado':'ever_married',
            'tipo_trabalho':'work_type','tipo_residencia':'residence_type',
            'fumante':'smoking_status'}, inplace=True)

#Avaliando os valores minimos, máximos, média e desvio padrão
for atb in ('age', 'gender', 'hypertension', 'heart_disease', 'ever_married',
            'work_type', 'residence_type', 'avg_glucose_level', 'bmi',
            'smoking_status', 'stroke'):
    print('Variavel {} minimo {} maximo {} media {} desvio padrao {}'
    .format(atb, df[atb].min(),
            df[atb].max(),
            df[atb].mean(),
            df[atb].std()))

"""# CORRELAÇÃO E COLINEARIDADE"""

import matplotlib.pyplot as plt
import seaborn as sns

df.head()

df.corr()

plt.figure(figsize=(12,7))
plt.title('Correlação entre os atributos', y=1.05, size=15)
sns.heatmap(df.astype(float).corr(), annot=True,cmap='YlGnBu')
```

```
# Detecting Multicollinearity with VIF - Python

from statsmodels.stats.outliers_influence import variance_inflation_factor

# the independent variables set
X = df[['age','gender', 'hypertension', 'heart_disease', 'ever_married',
        'work_type', 'residence_type', 'avg_glucose_level', 'bmi',
        'smoking_status', 'stroke']]

# VIF dataframe
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns

# calculating VIF for each feature
vif_data["VIF"] = [variance_inflation_factor(X.values, i)
                          for i in range(len(X.columns))]

print(vif_data)

"""# VISUALIZAÇÃO DOS DADOS"""

df['stroke'].value_counts()

df['gender'].value_counts()

df['hypertension'].value_counts()

df['ever_married'].value_counts()

df['work_type'].value_counts()

df['residence_type'].value_counts()

df['smoking_status'].value_counts()

df['heart_disease'].value_counts()

import matplotlib.pyplot as plt
plt.style.use('ggplot')
```

```
import seaborn as sns

#Balanceamento de dados

labels =df['stroke'].value_counts(sort = True).index
sizes = df['stroke'].value_counts(sort = True)

colors = ["lightblue","red"]
explode = (0.05,0)

plt.figure(figsize=(7,7))
plt.pie(sizes, explode=explode, labels=labels, colors=colors,
        autopct='%1.1f%%', shadow=True, startangle=90,)

plt.title('Casos de AVC antes do balanceamento')
plt.show()

plt.figure(figsize=(10,5))
sns.countplot(data=df,x='gender');

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(15, 5))
df.plot(kind='scatter', x='age', y='avg_glucose_level', alpha=0.5,
        color='blue', ax=axes[0], title="Idade vs. Nível de glicose")
df.plot(kind='scatter', x='bmi', y='avg_glucose_level', alpha=0.5,
        color='red', ax=axes[1], title="BMI vs. Nível de glicose")
plt.show()

strok=df.loc[df['stroke']==1]
plt.figure(figsize=(10,5))
sns.countplot(data=strok,x='heart_disease',palette='Reds');

plt.figure(figsize=(10,5))
sns.countplot(data=strok,x='hypertension',palette='Blues');

plt.figure(figsize=(10,5))
sns.countplot(data=strok,x='smoking_status',palette='Purples');

plt.figure(figsize=(12,10))
```

```
sns.distplot(df[df['stroke'] == 0]["bmi"], color='blue') # No Stroke
sns.distplot(df[df['stroke'] == 1]["bmi"], color='red') # Stroke

#plt.title('Sem AVC vs Com AVC por IMC', fontsize=15)
plt.xlim([10,100])
plt.show()

plt.figure(figsize=(12,10))
#No Stroke
sns.distplot(df[df['stroke'] == 0]["avg_glucose_level"], color='blue')
#Stroke
sns.distplot(df[df['stroke'] == 1]["avg_glucose_level"], color='red')

#plt.title('Sem AVC vs Com AVC por Glicemia', fontsize=15)
plt.xlim([30,330])
plt.show()

plt.figure(figsize=(12,10))

sns.distplot(df[df['stroke'] == 0]["age"], color='blue') # No Stroke
sns.distplot(df[df['stroke'] == 1]["age"], color='red') # Stroke

#plt.title('Sem AVC vs Com AVC por Idade', fontsize=15)
plt.xlim([18,100])
plt.show()

"""# TESTANDO OS MODELOS

# BALANCEAMENTO
"""

features = ['hypertension', 'heart_disease', 'avg_glucose_level', 'bmi',
            'gender','ever_married', 'work_type', 'residence_type',
            'smoking_status', 'age']

from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE

X , y = df[features],df["stroke"]
```

```
x_train,x_test,y_train,y_test = train_test_split(X,y,test_size=0.3,
random_state=23)

sm = SMOTE()

X_res, y_res = sm.fit_resample(x_train,y_train)

a = y_res
unique, counts = np.unique(a, return_counts=True)
ab = dict(zip(unique, counts))
ab

import matplotlib.pyplot as plt

# Data to plot
labels = []
sizes = []

for x, y in ab.items():
    labels.append(x)
    sizes.append(y)

# Plot
colors = ["lightblue","red"]
explode = (0.05,0)

plt.figure(figsize=(7,7))
plt.pie(sizes, explode=explode, labels=labels, colors=colors,
autopct='%1.1f%%', shadow=True, startangle=90,)

plt.title('Casos de AVC depois do balanceamento com SMOTE')
plt.show()

"""# MODELAGEM E TREINAMENTO"""

x_train = X_res
y_train = y_res
```

```
x_train

print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)

from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict
import xgboost as xgb

class ClassifierModel(object):
    def __init__(self, clf, params=None):
        self.clf = clf(**params)

    def train(self, x_train, y_train):
        self.clf.fit(x_train, y_train)

    def fit(self,x,y):
        return self.clf.fit(x,y)

    def feature_importances(self,x,y):
        return self.clf.fit(x,y).feature_importances_

    def predict(self, x):
        return self.clf.predict(x)

from sklearn.model_selection import cross_validate

def trainModel(model, x_train, y_train, x_test, n_folds):
    cv = KFold(n_splits= n_folds, random_state=0, shuffle=True)
    scoring = ['precision_macro', 'recall_macro']
```

```
    metrics = cross_validate(model.clf, x_train, y_train, cv=cv,
                scoring=scoring)
    cv_scores = cross_val_score(model.clf, x_train, y_train,
                scoring='accuracy', cv=cv, n_jobs=-1)
    y_pred = cross_val_predict(model.clf, x_train, y_train,
             cv=cv, n_jobs=-1)
    return cv_scores, y_pred, metrics

# Random Forest parameters
rf_params = {
    'n_estimators': 400,
    'max_depth': 5,
    'min_samples_leaf': 3,
    'max_features' : 'sqrt',
}
rfc_model = ClassifierModel(clf=RandomForestClassifier, params=rf_params)
rfc_scores, rfc_train_pred, rfc_metrics = trainModel(rfc_model,x_train,
                                                     y_train, x_test, 5)
rfc_scores

rfc_metrics

m1 = rfc_metrics['fit_time'].mean()
m2 = rfc_metrics['score_time'].mean()
m3 = rfc_metrics['test_precision_macro'].mean()
m4 = rfc_metrics['test_recall_macro'].mean()

m1, m2, m3, m4

# Extra Trees Parameters
et_params = {
    'n_jobs': -1,
    'n_estimators':400,
    'max_depth': 5,
    'min_samples_leaf': 2,
}
etc_model = ClassifierModel(clf=ExtraTreesClassifier, params=et_params)
etc_scores, etc_train_pred, etc_metrics = trainModel(etc_model,x_train,
                                                     y_train, x_test, 5)
```

```
etc_scores

etc_metrics

m1 = etc_metrics['fit_time'].mean()
m2 = etc_metrics['score_time'].mean()
m3 = etc_metrics['test_precision_macro'].mean()
m4 = etc_metrics['test_recall_macro'].mean()

m1, m2, m3, m4

# AdaBoost parameters
ada_params = {
    'n_estimators': 400,
    'learning_rate' : 0.65
}
ada_model = ClassifierModel(clf=AdaBoostClassifier, params=ada_params)
ada_scores, ada_train_pred, ada_metrics = trainModel(ada_model,x_train,
                                                     y_train, x_test, 5)
ada_scores

ada_metrics

m1 = ada_metrics['fit_time'].mean()
m2 = ada_metrics['score_time'].mean()
m3 = ada_metrics['test_precision_macro'].mean()
m4 = ada_metrics['test_recall_macro'].mean()

m1, m2, m3, m4

# Gradient Boosting parameters
gb_params = {
    'n_estimators': 400,
    'max_depth': 6,
}
gbc_model = ClassifierModel(clf=GradientBoostingClassifier,
                                                params=gb_params)
gbc_scores, gbc_train_pred, gbc_metrics = trainModel(gbc_model,x_train,
                                                     y_train, x_test, 5)
```

```
gbc_scores

gbc_metrics

m1 = gbc_metrics['fit_time'].mean()
m2 = gbc_metrics['score_time'].mean()
m3 = gbc_metrics['test_precision_macro'].mean()
m4 = gbc_metrics['test_recall_macro'].mean()

m1, m2, m3, m4

# Gaussian NB Classifier parameters
gnb_params = {
    'var_smoothing': 1e-09
    }
gnb_model = ClassifierModel(clf=GaussianNB, params=gnb_params)
gnb_scores, gnb_train_pred, gnb_metrics = trainModel(gnb_model,x_train,
                                                     y_train, x_test, 5)
gnb_scores

gnb_metrics

m1 = gnb_metrics['fit_time'].mean()
m2 = gnb_metrics['score_time'].mean()
m3 = gnb_metrics['test_precision_macro'].mean()
m4 = gnb_metrics['test_recall_macro'].mean()

m1, m2, m3, m4

from sklearn.neighbors import KNeighborsClassifier
# KNeighborsClassifier parameters
knc_params = {
    'n_neighbors': 5
    }
knc_model = ClassifierModel(clf=KNeighborsClassifier, params=knc_params)
knc_scores, knc_train_pred, knc_metrics = trainModel(knc_model,x_train,
                                                     y_train, x_test, 5)
knc_scores
```

```
knc_metrics

m1 = knc_metrics['fit_time'].mean()
m2 = knc_metrics['score_time'].mean()
m3 = knc_metrics['test_precision_macro'].mean()
m4 = knc_metrics['test_recall_macro'].mean()
m1, m2, m3, m4

acc_pred_train = pd.DataFrame ({
     'AdaBoost': ada_scores.ravel(),
     'RandomForest': rfc_scores.ravel(),
     'ExtraTrees': etc_scores.ravel(),
     'GaussianNB' : gnb_scores.ravel(),
     'K-NearestNeighbor' : knc_scores.ravel(),
     'GradientBoosting' : gbc_scores.ravel()
    })
acc_pred_train.head().mean()

plt.figure(figsize=(7,5))
plt.title('Correlação entre os classificadores', y=1.05, size=15)
sns.heatmap(acc_pred_train.astype(float).corr(), annot=True,cmap='YlGnBu')

"""# Stacking Model"""

x_train_return = np.column_stack(( ada_train_pred, rfc_train_pred,
etc_train_pred , gnb_train_pred, knc_train_pred, gbc_train_pred))

x_train_return.shape

def trainStackModel(x_train, y_train, x_test, n_folds, seed):
    cv = KFold(n_splits= n_folds, random_state=seed, shuffle=True)
    gbm = xgb.XGBClassifier(
     n_estimators= 2000,
     max_depth= 4,
     min_child_weight= 2,
     gamma=0.9,
     subsample=0.8,
     colsample_bytree=0.8,
     objective= 'binary:logistic',
```

```
     scale_pos_weight=1).fit(x_train, y_train)

    scores = cross_val_score(gbm, x_train, y_train, scoring='accuracy',
                                                                cv=cv)
    return scores

stackModel_scores = trainStackModel(x_train_return, y_train, x_test, 5, 0)

acc_pred_train

acc_pred_train['StackingModel'] = stackModel_scores
acc_pred_train

stackModel_scores.mean()

acc_pred_train.mean()
```

APÊNDICE B – Código fonte da API

```
import xgboost as xgb
from flask import Flask, request
import pandas as pd
import numpy as np
import flasgger
from flasgger import Swagger
import pickle

app = Flask(__name__)

swagger_config = {
    "headers": [
    ],
    "specs": [
        {
            "endpoint": 'apispec_1',
            "route": '/apispec_1.json',
            "rule_filter": lambda rule: True,
            "model_filter": lambda tag: True,
        }
    ],
    "static_url_path": "/flasgger_static",
    "swagger_ui": True,
    "specs_route": "/apidocs/"
}

template = {
    "swagger": "2.0",
    "info": {
        "title": "Predição de AVC",
        "description": "API desenvolvida por Gabriela Campos Gama para
                        o trabalho de conclusão do curso de Engenharia
                        da Computaçao - CEFET/MG",
        "contact": {
            "responsibleOrganization": "ME",
```

```
            "responsibleDeveloper": "Me",
            "email": "camposgamagabriela@gmail.com",
        },
        "termsOfService": "http://me.com/terms",
        "version": "0.0.1"
    }
}

Swagger(app, config=swagger_config, template=template)
# Adaboost
pickle_in = open('models/model_ab.pkl', 'rb')
model_ab = pickle.load(pickle_in)
# Extra Trees
pickle_in = open('models/model_et.pkl', 'rb')
model_et = pickle.load(pickle_in)
# Gradient Boosting
pickle_in = open('models/model_gbc.pkl', 'rb')
model_gbc = pickle.load(pickle_in)
# GaussinNB
pickle_in = open('models/model_gnb.pkl', 'rb')
model_gnb = pickle.load(pickle_in)
# KNN
pickle_in = open('models/model_knn.pkl', 'rb')
model_knn = pickle.load(pickle_in)
# Random Forest
pickle_in = open('models/model_rf.pkl', 'rb')
model_rf = pickle.load(pickle_in)
# XGBoost
pickle_in = open('models/model_xgb.pkl', 'rb')
model_xgb = pickle.load(pickle_in)


@app.route('/')
def welcome():
    return 'Deploy do modelo de ML'


@app.route('/predicao_parametros', methods=["Get"])
def predict_stroke_parameters():
```

```
    """Predição utilizando os parâmetros
    ---
    tags:
          - Predição
    parameters:
        - name: idade
          in: query
          type: number
          required: true
        - name: genero
          in: query
          type: number
          required: true
        - name: hipertensao
          in: query
          type: number
          required: true
        - name: doenca_do_coracao
          in: query
          type: number
          required: true
        - name: ja_se_casou
          in: query
          type: number
          required: true
        - name: tipo_trabalho
          in: query
          type: number
          required: true
        - name: tipo_residencia
          in: query
          type: number
          required: true
        - name: nivel_glicose
          in: query
          type: number
          required: true
        - name: imc
          in: query
```

```
        type: number
        required: true
      - name: condicao_fumante
        in: query
        type: number
        required: true
responses:
    200:
        description: The output values
"""

age = request.args.get("idade")
gender = request.args.get("genero")
hypertension = request.args.get("hipertensao")
heart_disease = request.args.get("doenca_do_coracao")
ever_married = request.args.get("ja_se_casou")
work_type = request.args.get("tipo_trabalho")
residence_type = request.args.get("tipo_residencia")
avg_glucose_level = request.args.get("nivel_glicose")
bmi = request.args.get("imc")
smoking_status = request.args.get("condicao_fumante")

predictions = [
    model_ab.predict(
        [pd.to_numeric([hypertension, heart_disease, avg_glucose_level,
        bmi, gender, ever_married, work_type, residence_type,
        smoking_status, age])]),
    model_et.predict(
        [pd.to_numeric([hypertension, heart_disease, avg_glucose_level,
        bmi, gender, ever_married, work_type, residence_type,
        smoking_status, age])]),
    model_gbc.predict(
        [pd.to_numeric([hypertension, heart_disease, avg_glucose_level,
        bmi, gender, ever_married, work_type, residence_type,
        smoking_status, age])]),
    model_gnb.predict(
        [pd.to_numeric([hypertension, heart_disease, avg_glucose_level,
        bmi, gender, ever_married, work_type, residence_type,
        smoking_status, age])]),
```

```
        model_knn.predict(
            [pd.to_numeric([hypertension, heart_disease, avg_glucose_level,
            bmi, gender, ever_married, work_type, residence_type,
            smoking_status, age])]),
        model_rf.predict(
            [pd.to_numeric([hypertension, heart_disease, avg_glucose_level,
            bmi, gender, ever_married, work_type, residence_type,
            smoking_status, age])])

    predictions_concat = np.concatenate(predictions)
    print(predictions_concat)
    result = model_xgb.predict([predictions_concat])

    return str(result[0])


if __name__ == '__main__':
    app.debug = True
    app.run(host='0.0.0.0', port=5080)
```