LUCAS EDUARDO SCHOENFELDER

PREDIÇÃO DE EPÍTOPOS LINEARES DE CÉLULAS B: UMA ABORDAGEM POR *ENSEMBLES*

(*versão pré-defesa, compilada em 19 de setembro de 2022*)

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Informática Biomédica, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Informática Biomédica*.

Orientador: Eduardo Jaques Spinosa.

CURITIBA PR

2022

RESUMO

O design de vacinas baseado em epítopos têm ganhado atenção desde que a primeira vacina desenvolvida por este método foi criada em 1981. Por este método, vacinas contra agentes infecciosos podem ser desenvolvidas mais rápido, por um custo menor e com menos efeitos adversos. Devido a isso, vários métodos computacionais foram desenvolvidos para a predição de epítopos de células B, porém a performance destes ainda deixa a desejar. Uma possível forma de melhorar a acurácia é utilizando *ensembles* de classificadores. Neste trabalho, analisou-se o desempenho de quatro modelos de *ensembles* baseados em árvores de decisão para a tarefa de classificação binária de epítopos lineares de células B. Os algoritmos Adaboost, *Extra Trees*, *Gradient Boosting* e *eXtreme Gradient Boosting* (XGB) foram treinados em dois *datasets* de epítopos virais, um desbalanceado e com sequências de aminoácidos de tamanho variado (viral) e outro com classes balanceadas e sequências de tamanho fixo (DeepVacPred). Cada algoritmo passou por uma etapa de otimização de hiper parâmetros utilizando o método *GridSearch* com validação cruzada de 5 *folds*, e os desempenhos foram medidos utilizando o coeficiente de correlação de Matthews (MCC). Identificou-se que a performance dos *ensembles* varia de acordo com as características do *dataset* de treino. No *dataset* viral, o algoritmo *Extra Trees* foi o melhor método com um MCC de 0.655, enquanto que no DeepVacPred o XGB se destacou com um MCC de 0.634.

Palavras-chave: Epítopo de células B. Design de vacinas. Aprendizado por *ensemble*

ABSTRACT

Epitope-based vaccine design has received growing interest since the first epitope-based vaccine was developed in 1981. With this method, vaccines against infectious agents can be developed faster and with less associated costs and adverse reactions. Thus, many computational methods were developed for the prediction of B-cell epitopes. However, their performance is lacking. One way of improving their accuracy is utilizing classifier ensembles. In this study, the performance of four decision tree-based ensemble models were analyzed in the task of binary classification of linear B-cell epitopes. The Adaboost, Extra Trees, Gradient Boosting and eXtreme Gradient Boosting (XGB) algorithms were trained using two datasets of viral epitopes, one imbalanced and with varying aminoacid sequence length (viral) and the other balanced and with fixed-length sequences (DeepVacPred). Each algorithm had their hyperparameters optimized by the GridSearch method utilizing 5-fold cross-validation, and their performance was measured by the Matthews correlation coefficient (MCC). The ensembles performed differently for each dataset. On the viral dataset, the Extra Trees algorithm performed better, achieving an MCC score of 0.655. On the DeepVacPred dataset, XGB was the superior method with an MCC score of 0.634.

Keywords: B-Cell epitope. Vaccine design. Ensemble Learning.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Com exceção da descoberta do tratamento da água para produção de água potável, nenhuma outra intervenção humana teve um impacto tão grande no crescimento populacional e na redução da mortalidade quanto a vacinação (Mortimer e Plotkin, 1998). Dentre os casos de sucessos dessa invenção, a erradicação do vírus da varíola em 1980, agente causador de uma das doenças mais devastadoras na história da humanidade, continua sendo considerado como um dos sucesso mais profundos e notáveis na história da saúde pública, e espera-se que esse sucesso logo se repita com o vírus da poliomelite, que hoje falta ser erradicado apenas no continente asiático. De fato, de acordo com Parvizpour et al., "a vacina é um dos fármacos mais aplicáveis e com um dos melhores custo-benefício, e têm sido usada para a prevenção e tratamento de doenças por mais de dois séculos"(2020, p. 1034, tradução do autor).

Ainda assim, diferentes epidemias como a de Ebola em 2014 e a de Zika em 2015, e mais recentemente a pandemia de COVID-19 revelaram que a sociedade moderna continua vulnerável e despreparada para enfrentar doenças altamente infecciosas, e isso se deve em parte pelo longo tempo necessário para sintetizar vacinas contra novos patógenos uma vez que estes sejam identificados (Kayser e Ramzan, 2021). Métodos tradicionais de desenvolvimento de vacinas podem levar entre 5 a 15 anos, com custos imensos associados e, portanto, um dos maiores desafios na era pós genômica é a determinação de regiões antigênicas (ou epítopos) para o aperfeiçoamento da simulação da resposta imune humana (Parvizpour et al., 2020).

O estudo do design de vacinas através da predição de epítopos de células B não é recente, tendo a primeira vacina baseada em epítopos sendo desenvolvida em 1981 (Nain et al., 2020). Métodos iniciais eram baseados em escalas de aminoácidos, e a má performance destes levou a um interesse pela aplicação de métodos de aprendizado de máquina ao problema (Sanchez-Trincado et al., 2017). No entanto, décadas depois, a performance dos sistemas disponíveis para a predição de epítopos ainda deixa a desejar (Liu et al., 2020). Frente a isso, uma das possíveis soluções para o problema é a utilização de *ensembles*, técnicas que utilizam vários algoritmos de aprendizado de máquina trabalhando de forma conjunta para tentar obter performances que um único algoritmo não conseguiria. Como diz Parvizpour et al. (2020): "[...] neste cenário, é imperativo que se estude o uso de procedimentos multi-métodos que incorporem o resultado de vários preditores para a predição de epítopos de células B".

Neste sentido, este trabalho propõe implementar e analisar a performance de diferentes modelos de *ensemble* para a tarefa de classificação binária de epítopos lineares de células B. Para isso, propôs-se investigar a performance de três algoritmos de *ensemble* baseados em árvores de decisão, *Adaboost*, *Gradient Boosting* e *ExtraTrees*, em dois *datasets* de epítopos virais de células B.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Este trabalho tem como objetivo implementar um modelo de classificação binária de epítopos lineares através de *ensembles* de classificadores. Para isso, dois *datasets* de epítopos virais de células B foram coletados para treino e validação dos modelos, um com classes balanceadas e sequências de tamanho fixo de aminoácidos e outro com classes desbalanceadas e sequências de tamanho variável. Para investigar o comportamento dos modelos, todo classificador passou por uma etapa de otimização de hiper parâmetros utilizando o método *gridsearch* com validação cruzada de cinco *folds*.

1.1.2 Objetivos Específicos

- Adquirir um *dataset* desbalanceado com sequências de tamanho variável de aminoácidos classificados em epítopos virais e não epítopos.
- Adquirir um *dataset* balanceado de sequências de tamanho fixo de aminoácidos classificados em epítopos virais e não epítopos.
- Implementar classificadores baseados em *ensembles* e treiná-los utilizando validação cruzada para encontrar os hiper parâmetros ótimos de cada modelo.
- Analisar os resultados e determinar qual algoritmo propõe a melhor solução para o problema

2 FUNDAMENTAÇÃO TEÓRICA

2.1 BASE BIOLÓGICA

2.1.1 Antígenos e Epítopos

Um antígeno (Ag) é uma substância estranha a um organismo e que possui a capacidade de gerar uma resposta imune quando reconhecida pelo sistema imunológico do organismo invadido. Sua origem pode ser exógena, proveniente de um patógeno como uma bactéria, vírus, fungo ou parasita; ou endógena, quando células do organismo apresentam proteínas do complexo principal de histocompatibilidade (MHC) devido a uma infecção viral ou a proteínas produzidas por uma célula tumoral (Parvizpour et al., 2020; Manavalan et al., 2018). Antígenos podem ter naturezas químicas diferentes, porém a maior parte dos antígenos são proteínas, e por isso a predição de epítopos costuma focar neste tipo de antígeno (Sanchez-Trincado et al., 2017).

Frente à presença de um antígeno, o sistema imune pode gerar respostas humorais ou celulares. A resposta humoral é articulada por células B através da produção e e secreção de anticorpos (Ab) que reconhecem e se ligam ao antígeno enquanto este está no ambiente extracelular. Uma vez que um antígeno consiga adentrar uma célula do organismo hospedeiro, a resposta imune celular é ativada e orquestrada por células T, cujo intuito é a ativação de macrófagos que eliminem o antígeno através da fagocitose ou através da eliminação da célula infectada (Mutneja et al., 2014; Abbas et al., 2021; Parvizpour et al., 2020; Schatzmann e Pereira, 2021). Neste trabalho, focou-se na resposta humoral ativada pela interação entre antígenos e células B.

A região de um antígeno à qual um anticorpo (ou receptores de células T) se liga recebe o nome de epítopo ou determinante antigênico. Epítopos de células B podem ser classificados de forma abrangente em dois tipos: epítopos lineares, que consistem em uma sequência contígua de aminoácidos do antígeno que é reconhecida pela sequência cognata do anticorpo, e epítopos conformacionais, formados por aminoácidos potencialmente separados na sequência proteica do antígeno, mas unidos na estrutura tridimensional decorrente do dobramento da proteína. A diferença entre os dois tipos de epítopos está esquematizada na figura 2.1. Estima-se que cerca de 90% dos epítopos de células B são conformacionais e apenas 10% são lineares, porém mesmo epítopos conformacionais podem conter grupos de aminoácidos contíguos e que também são contíguos na estrutura 3D da proteína, portanto a distinção entre os dois tipos nem sempre é clara (Galanis et al., 2021).

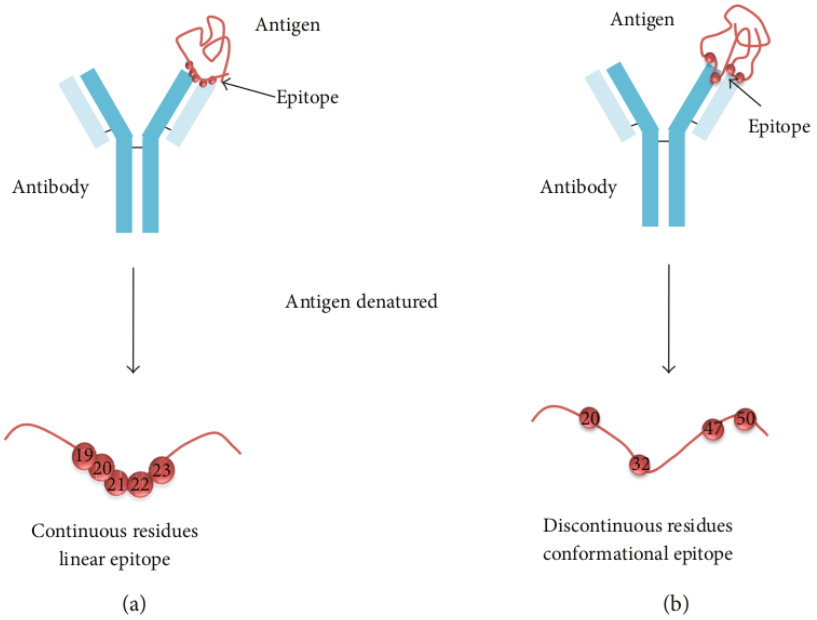

Figura 2.1: Epítopos lineares e conformacionais de células B. Epítopos lineares (a) são compostos por resíduos (aminoácidos) contínuos, enquanto epítopos conformacionais (b) contém resíduos espalhados pela sequência do antígeno. FONTE: Sanchez-Trincado et al. (2017).

2.1.2 Resposta Imune

O sistema imunológico tem como função proteger o organismo de substâncias estranhas e agentes infecciosos. Para isso, dois tipos de imunidade agem em conjunto para orquestrar a resposta do sistema: a imunidade inata e a adaptativa.

A imunidade inata é relacionada a mecanismos de defesa não específicos, que protegem o organismo de infecções por patógenos mesmo sem ter algum contato prévio com o mesmo. Todos os organismos multicelulares possuem alguma forma de imunidade inata (Sanchez-Trincado et al., 2017). No caso de humanos, a própria pele e a resposta inflamatória que protege a mesma em casos de ruptura do tecido epitelial fornece uma barreira não-específica que protege contra inúmeras doenças (Schatzmann e Pereira, 2021; Abbas et al., 2021). Os elementos que determinam a resposta da imunidade inata são vários, podendo ser naturais a uma espécie - como é o caso de cães e gatos, que possuem resistência ao vírus Influenza A sendo que o mesmo é altamente infeccioso entre aves, suínos e equinos (Machado e Machado, 1992); ou específicos daquele hospedeiro, como é o caso de variantes genéticas hereditárias como a mutação do gene *SLC30A8*, a qual torna o indivíduo 65% menos propenso a desenvolver o quadro de diabetes mesmo com a existência de fatores de risco (Williams, 2016).

A imunidade adaptativa, em contrapartida, só está presente em vertebrados, é altamente específica e depende de uma exposição prévia a um patógeno para o desenvolvimento da memória imunológica (Sanchez-Trincado et al., 2017; Galanis et al., 2021). Ela é articulada pelos linfócitos, as células B e T, responsáveis pela resposta humoral e pela resposta celular, respectivamente. Na resposta humoral, imunoglobulinas ligadas à membrana de células B atuam como receptores de antígenos e, após a ligação de um antígeno, estas se diferenciam e secretam formas solúveis de

imunoglobulinas conhecidas como anticorpos, que possuem como função neutralizar patógenos e marcá-los para subsequente destruição. Já a resposta celular ocorre quando células T reconhecem e se ligam a moléculas do complexo principal de histocompatibilidade (MHC) presentes em células apresentadoras de antígenos (APCs). Células T podem ser auxiliares ou citotóxicas, e possuem a função de ativar macrófagos para eliminar o antígeno ou de destruir a célula infectada por inteiro, respectivamente (Schatzmann e Pereira, 2021; Sanchez-Trincado et al., 2017; Abbas et al., 2021). As funções de cada uma das células descritas estão representadas na figura 2.2.

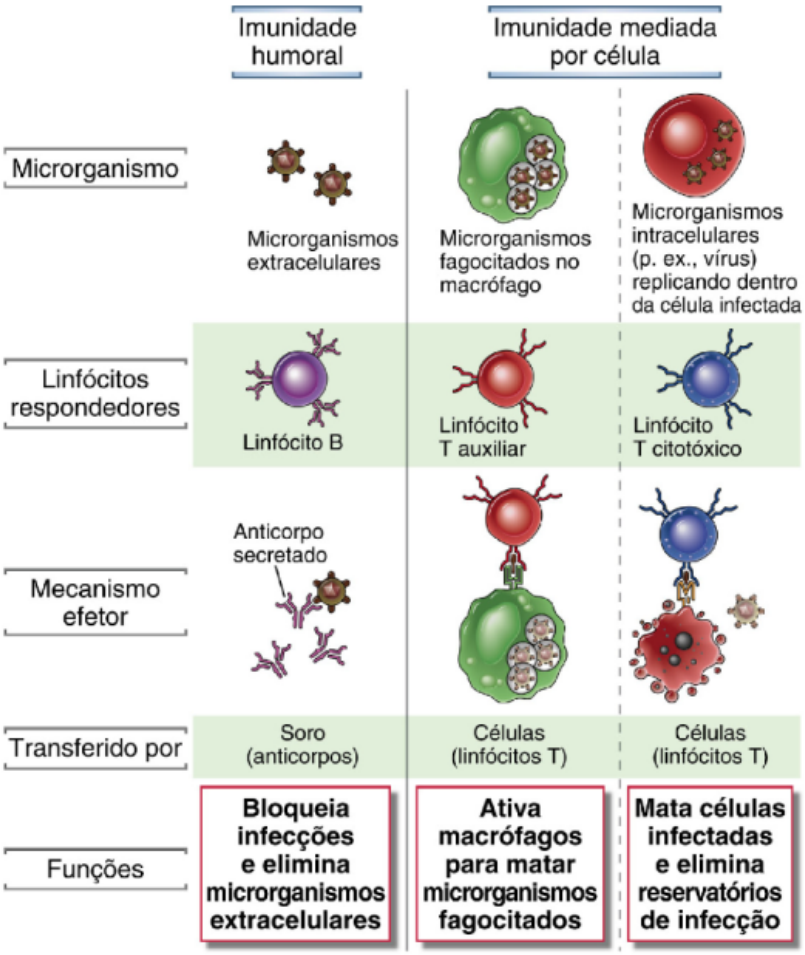

Figura 2.2: Função e linfócito responsável por cada uma das possíveis respostas do sistema imune adaptativo. FONTE: Abbas et al. (2021)

2.1.3 Design de Vacinas baseado em Epítopos

A vacina é uma preparação biológica que estimula a produção de anticorpos para induzir uma resposta imune a uma doença específica, e a imunização que elas fornecem induz as imunidades humoral e celular, resultando em um decréscimo substancial nas taxas de mortalidade e de morbidade (taxa de portadores de uma determinada doença em relação à população total) e na melhoria da expectativa de vida humana (Parvizpour et al., 2020; Zhang et al., 2015).

Embora existam há séculos e tenham tido avanços incríveis nas últimas décadas, não existe uma estratégia única para a implementação de abordagens para o design de vacinas (Parvizpour et al., 2020). Ainda assim, e visto que a principal consideração no design de vacinas é a redução do impacto da resposta imune induzida sem redução da capacidade de indução da resposta imune em si (Zhang et al., 2015), o design de vacinas baseado em epítopos tem ganhado um grande interesse recentemente por oferecer várias vantagens, como a eliminação de respostas imunes indesejadas pelo design de construções específicas, a geração de uma imunidade prolongada com as respostas imunes requeridas e pela eficiência tanto em termos de tempo quanto de custo (Parvizpour et al., 2020).

Sendo assim, podemos afirmar que o conceito chave das vacinas baseadas em epítopo é a estimulação da resposta imune através do design de moléculas que consigam imitar a estrutura e função de um verdadeiro epítopo, seja este de células B ou de células T, pois essa interação epítopo-linfócitos é a chave da imunidade adaptativa e, além disso, um epítopo é a menor molécula capaz de elicitar uma resposta imune humoral potente sem que haja efeitos colaterais perigosos para o corpo humano (Sun et al., 2013; Parvizpour et al., 2020; Galanis et al., 2021).

Métodos clássicos para a identificação de epítopos por imunologistas costumam ser caros, demandar muito tempo e nem sempre produzem resultados (Lo et al., 2021). Neste sentido, métodos *in silico* para a identificação e predição de epítopos que apresentem boa acurácia são de imenso interesse para o desenvolvimento de novas vacinas e de fármacos contra patógenos virais comuns como o HIV e os vírus de hepatite e influenza, uma vez que, com estes métodos, imunologistas podem focar em estudar apenas os segmentos de proteínas com alta probabilidade de serem antigênicos, reduzindo a complexidade, o tempo despendido e os custos dos experimentos (Lo et al., 2021; Bahai et al., 2021). As diferentes etapas envolvidas no processo de design de vacinas baseadas em epítopos estão resumidas na figura 2.3.

2.2 BASE COMPUTACIONAL

2.2.1 Aprendizado de Máquina

Algoritmos de aprendizado de máquina visam aprender padrões nos dados de entrada utilizando métodos estatísticos e matemáticos para realizar alguma tarefa para a qual uma solução algorítmica convencional seria muito complexa ou até mesmo impossível de se desenvolver. Eles são classificados de forma geral em dois tipos, supervisionado e não supervisionado, de acordo com a presença ou ausência de rótulos nos dados de treino.

A classificação de epítopos de células B é uma aplicação do aprendizado de máquina supervisionado, mais especificamente da classificação binária. Neste problema, os dados de entrada são sequências de aminoácidos classificadas em duas classes, epítopos e não epítopos, e os algoritmos aplicados visam aprender quais são as características que diferenciam essas duas classes de forma a poder classificar sequências que o modelo de aprendizado de máquina não tenha visto antes.

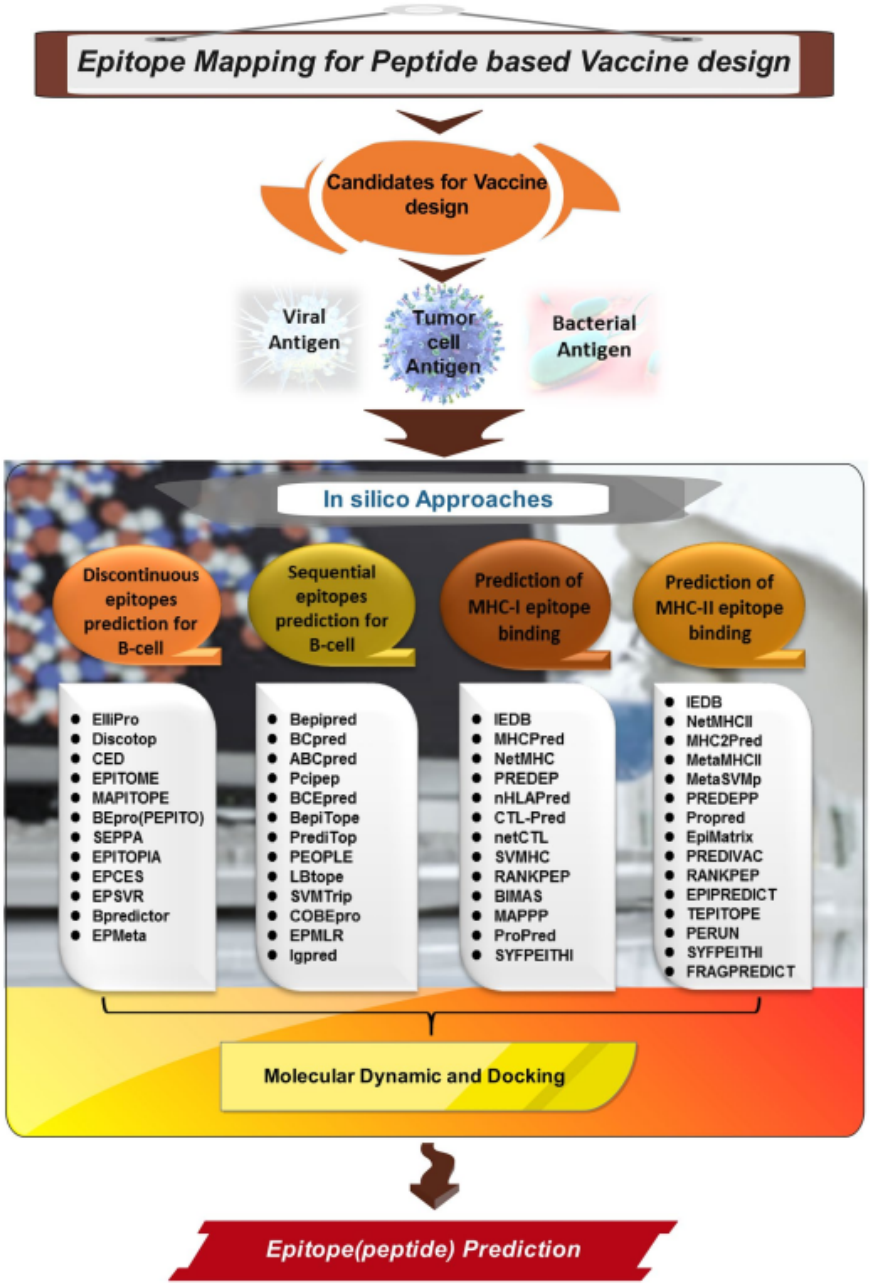

Figura 2.3: Diagrama das etapas do desenvolvimento de vacinas baseadas em epítopos. Em um primeiro momento, antígenos candidatos são selecionados, podendo ser de origem bacterial, viral ou de células tumorais. Depois, métodos *in silico* são aplicados para encontrar os epítopos de células B (lineares e conformacionais) e de células T (MHC-I ligantes e MHC-II ligantes). Então, para avaliar a interação entre os epítopos com melhor score de predição e os sítios de ligação mais favoráveis, são utilizados estudos de dinâmica molecular e de docagem molecular. FONTE: Raoufi et al. (2020)

2.2.2 *Ensembles*

No aprendizado por *ensemble* utiliza-se uma combinação de algoritmos de aprendizado de máquina (que nesse contexto são chamados de modelos base) trabalhando juntos para formar um modelo mais complexo que, espera-se, tenha uma performance melhor do que a dos modelos base que o compõem. *Ensembles* podem incorporar vários modelos que implementam o

mesmo algoritmo de aprendizado (*ensemble* homogêneo) ou modelos que utilizam algoritmos de aprendizado diferentes (*ensemble* heterogêneo).

A forma de agregar os modelos base utilizados em um *ensemble* difere de acordo com o propósito pelo qual se está utilizando aquela combinação. De forma geral, podemos identificar três técnicas amplamente usadas no aprendizado por *ensemble*:

- **Bagging**: Também conhecida como *bootstrap aggregating* e comumente usada em *ensembles* homogêneos, esta técnica consiste em agregar múltiplas versões de um modelo preditivo em que cada versão é treinada de forma independente e utilizando um subconjunto único do *dataset* de treino. Como cada modelo aprende um nicho dos dados e a predição final do *ensemble* é obtida pela média das predições de cada modelo base, espera-se que o *ensemble* treinado com essa técnica possua menos variância (variabilidade das predições de um modelo ao utilizar porções diferentes do *dataset* de treino) do que os modelos individuais que o compõem.

- **Boosting**: A ideia nesta técnica, que comumente é utilizada com *ensembles* homogêneos, é criar um modelo sequencial em que cada modelo base, com exceção do primeiro, tem como função reduzir os erros do modelo anterior. Isso é feito com a intenção de reduzir o viés (erros nas predições que ocorrem quando o algoritmo aprende padrões muito específicos aos dados de treino, os quais não generalizam bem para dados reais daquela classe) que surge quando um único modelo base não consegue identificar padrões relevantes nos dados de treino. Este procedimento ocorre até que todos os dados de treino sejam aprendidos ou até que o número máximo de modelos base permitidos tenham sido inclusos.

- **Stacking**: Ou generalização empilhada, consiste em utilizar um algoritmo de aprendizado de máquina (que nesse contexto geralmente é chamado de meta-modelo) para aprender qual a melhor forma de combinar os votos de vários modelos base visando maximizar o poder preditivo de um *ensemble*. Comumente os *ensembles* neste caso costumam ser heterogêneos.

Diferentes esquemas de votos são utilizados para gerar uma predição final em um modelo de *ensemble*. Quando trabalha-se com um *ensemble* heterogêneo em que os modelos base apresentam performance similar, por exemplo, é possível utilizar o esquema *Voting Ensemble*, no qual duas configurações são permitas. Na configuração *hard voting*, cada modelo base tem como saída um valor binário, o voto daquele modelo para a classe predita, e a predição final do *ensemble* é obtida somando os votos de cada modelo e escolhendo a classe que obteve mais votos. Já na configuração *soft voting*, cada modelo base informa as probabilidades com as quais classificaria um exemplo em ambas as classes, e o *ensemble* tira a média de todas as probabilidades informadas para informar a classe final, aquela que teve a maior probabilidade cumulativa.

2.2.3 Árvores de Decisão

Ensembles para classificação costumam incorporar a ideia de juntar vários classificadores fracos, isto é, classificadores que têm resultados ligeiramente melhores que uma classificação aleatória, para criar um modelo robusto e com bom desempenho. Para isso, um algoritmo muito usado para compor os classificadores fracos é a árvore de decisão.

Uma árvore no contexto computacional é uma estrutura de dados formada por conjuntos de elementos que contém uma informação, os nós, e ligações direcionadas entre eles, as arestas. Dessa forma, uma árvore pode ser vista como um fluxograma, em que o ponto de partida, a raiz da árvore, é um nó a partir do qual qualquer outro pode ser alcançado e a direção das ligações estabelece uma relação hierárquica entre todos os nós da árvore. Se um nó possui ligações de saída (comumente referidos como "filhos"), ele recebe o nome de nó interno. Caso contrário, se um nó possui apenas arestas de entrada e nenhuma de saída, ele recebe o nome de folha ou nó terminal.

Em uma árvore de decisão para classificação, os nós internos representam regras de decisão e as folhas representam uma classificação. A ideia básica é iniciar da raiz e, a cada nível, decidir para qual filho ir com base na regra de decisão presente naquele nível. Ao chegar em uma folha, basta olhar o valor presente nela para determinar a classificação final. Um exemplo de uma árvore de decisão está presente na 2.4. Em termos técnicos, dado um *dataset* de treino e um conjunto de atributos que descrevem cada exemplo, uma árvore de decisão pode ser expressada como uma partição recursiva do espaço de instâncias (o domínio dos atributos utilizados). No caso de uma árvore de decisão binária, cada nó interno divide o espaço de instâncias em dois subespaços de acordo com uma função discreta aplicada nos valores de um dos atributos do vetor de característica, e cada folha é associada a uma classe (Rokach e Maimon, 2005).

O algoritmo se inicia com a criação da raiz. Para isso, observa-se o conjunto de características que descrevem os dados de treino. Para cada característica, calcula-se o valor de corte, isto é, o valor daquela característica que melhor separa os dados. Para isso, uma série de critérios podem ser usados, com um deles sendo a impureza de Gini. Ela é calculada da seguinte forma:

$$Gini = 1 - \sum_{i}^{N} \mu_i^2 \tag{2.1}$$

Em que N é o número de observações no *dataset* de treino e μ_i é a probabilidade da observação i ser incorretamente classificada usando certo valor de corte para uma dada característica (Baby et al., 2021). O índice de Gini varia entre 0 e 1, com 0 significando que todas as observações divididas por um valor de corte pertencem à mesma classe e estão corretamente classificados (uma divisão pura) e 1 significa que nenhuma observação foi corretamente classificada utilizando aquele valor de corte.

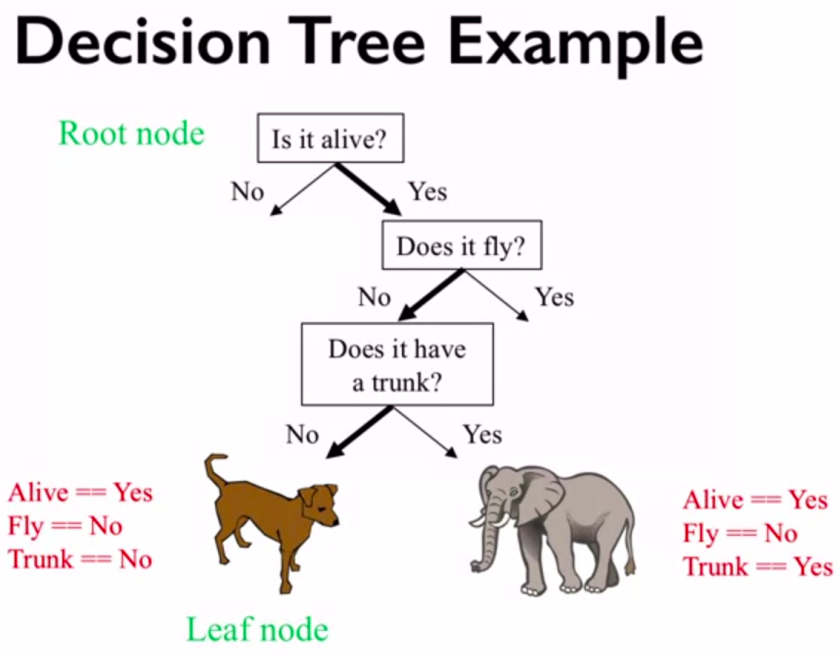

Figura 2.4: Exemplo de regras aprendidas por uma árvore de decisão. Nos nós folhas, está demonstrado o caminho percorrido para chegar até aquele nó. FONTE: Kovenko (2020)

Selecionado um valor de corte, cria-se a raiz da árvore e retira-se a característica usada para gerar esse nó do vetor de características a ser analisado no próximo passo. Nesse momento, como o algoritmo é recursivo, inicia-se o processo de novo.

O algoritmo continua dessa forma gerando nós internos e consequentemente *splits* nos dados, até que um dos critérios de parada sejam encontrados. Isto é, a árvore continua a crescer até que: não haja mais características para gerar um novo split; até que uma folha gerada possua índice de gini igual a 0, ou até que a profundidade da árvore exceda algum valor arbitrário delimitado por parâmetros do algoritmo.

Árvores de decisão são simples de implementar, não requerem tempos longos de treino nem etapas complexas de preprocessamento dos dados (Sanchez-Trincado et al., 2008). Além disso, o modelo final obtido é intuitivo de se entender e fornece *insights* sobre a hierarquia de importância das características utilizadas para descrever os dados. Dentre problemas deste método, podemos citar o fato de que, caso permita-se que as árvores possuam muitos nós internos (em outros palavras, caso permita-se a criação de modelos complexos), pode acontecer de o modelo final apresentar bons resultados porque ele aprendeu padrões muito específicos aos dados de treino, e que não generalizam bem para dados inéditos, problema que recebe o nome de *overfitting*.

Este comportamento, no entanto, pode ser controlado em métodos de *ensemble* impondo a limitação de que cada árvore utilizada só possa crescer até certo ponto. Dessa forma, limitando que cada árvore terá baixa complexidade e, consequentemente, menor poder preditivo, e utilizando várias árvores pode-se obter um modelo robusto através do princípio da sabedoria de multidões,

que prega que um grupo suficientemente grande de modelos com menor performance de predição agindo em conjunto irá superar os resultados de um único modelo de alta complexidade.

2.2.4 AdaBoost

O *Adaptive Boosting* é um algoritmo criado em 1995 por Yoav Freund e Robert Schapire que, cujo nome sugere, utiliza a técnica de *boosting* para criar um classificador forte através da combinação iterativa de vários classificadores fracos (tipicamente árvores de decisão com um único nível, também chamadas de *decision stumps*) treinados em subconjuntos dos dados de treino, em que a probabilidade de uma observação ser amostrada para compor um subconjunto é determinada por um peso atribuído a cada observação e atualizado de forma a tornar mais provável que observações difíceis de aprender sejam amostradas. De acordo com seus resultados individuais cada classificador recebe um poder de voto, de forma que ao final do treino uma predição é feita baseada na agregação dos votos de cada classificador fraco levando em conta o poder de voto destes (Freund e Schapire, 1997).

Considerando um *dataset* com N observações, o algoritmo inicia com um vetor de pesos iguais para cada observação. Como a soma dos valores do vetor de pesos sempre será 1, o peso de uma observação determina sua relevância no *dataset* de treino, sendo que quanto maior for o peso, maior será a probabilidade de aquela observação ser amostrada. Inicialmente, todos os pesos tem o mesmo valor w:

$$w_i = 1/N \in [0, 1] \tag{2.2}$$

Então, o algoritmo irá repetir os próximos passos M vezes, em que M é o número total de classificadores permitidos no modelo. Inicialmente, um subconjunto dos dados é amostrado de forma aleatória com respeito ao peso associado à cada observação. Inicializa-se um classificador G_m que é treinado neste subconjunto, e calcula-se o erro Err_m dividindo o total de classificações erradas pelo tamanho do subconjunto de treino. Em seguida, calcula-se a influência α do classificador utilizando a equação 2.3

$$\alpha_m = v \ln \frac{(1 - Err_m)}{Err_m} \tag{2.3}$$

O valor de α determina o poder de voto que o classificador G_m terá na predição final do Adaboost. Se um classificador apresenta poucos erros, seu valor de α será um valor relativamente grande e positivo. Analogamente, se o classificador acerta tanto quanto uma classificação aleatória (se seu $Err_m = 0.5$), o valor de α é zero e se o classificador apresenta muitos erros, seu α será um valor relativamente grande e negativo. O parâmetro v refere-se à taxa de aprendizado do algoritmo, usado para se ter um controle mais fino da contribuição de cada classificador.

Após o cálculo da influência, os pesos de cada observação no *dataset* de treino são atualizados de acordo com a equação 2.4:

$$w_i = w_{i-1} * \begin{cases} e^{-\alpha} & \text{se a observação i foi corretamente classificada} \\ e^{\alpha} & \text{se a observação i foi incorretamente classificada} \end{cases} \tag{2.4}$$

Em que i é o índice do peso w_i no vetor de pesos, e e é o número de Euler elevado a mais ou menos α. O mais ou menos da equação se refere a se a observação i foi corretamente classificada ou não pelo classificador G_m. Dessa forma, caso tenha sido incorretamente classificada, uma observação tem seu peso w aumentado, com o efeito de que observações difíceis de serem aprendidas pelos classificadores já adicionados ao modelo tenham maior probabilidade de serem amostradas pelo classificador adicionado na próxima iteração e consequentemente o modelo final tenha maior probabilidade de aprendê-las (Schapire, 2013).

Esse procedimento é repetido M vezes ou até que todas as observações tenham sido classificadas corretamente. Ao final do processo, uma nova observação x pode ser predita computando a hipótese $H(x)$ de acordo com a equação 2.5:

$$H(x) = \text{sign}(\sum_{i=1}^{M} \alpha_i G_m(x)) \tag{2.5}$$

Em que $H(x)$ é a classe predita e sign é a função sinal aplicada ao somatório da predição de cada classificador para a observação x multiplicada pela influência de cada classificador.

2.2.5 *Gradient Boosting Classifier*

Em 1997, Leo Breiman reinterpretou o algoritmo Adaboost através de uma análise estatística, em que ele percebeu que a técnica de *boosting* pode ser generalizada como um problema de otimização cujo objetivo é minimizar o erro do modelo através da adição de classificadores fracos de uma maneira parecida com o procedimento de gradiente descendente (Breiman, 1998). Utilizando-se dessa intuição, Friedman (2001) desenvolveu o primeiro modelo de *gradient boosting*, inicialmente apenas para o problema de regressão. Desde então, o modelo foi iterado por inúmeros pesquisadores de forma a torná-lo aplicável a diferentes problemas na área de aprendizado de máquina.

Sendo assim, podemos pensar no *gradient boosting classifier* como uma extensão do modelo AdaBoost, em que o *adaptive boosting* é combinado com minimização de pesos para criar um modelo mais robusto. Para isso, o *gradient boosting* utiliza três elementos: uma função de perda derivável, que tem como propósito estimar quão bom o modelo é na tarefa que ele se propõe a resolver; classificadores fracos, que comumente são árvores de decisão, e um modelo aditivo em que os classificadores fracos são adicionados um por vez, com seus hiper parâmetros configurados de forma a garantir que após a adição de um classificador o modelo final se aproxima do gradiente descendente da função de perda.

O algoritmo de *gradient boosting* recebe um *dataset* de tamanho N e uma função de perda. Neste trabalho utilizou-se a função *log loss*, cuja equação é:

$$L = -(y_i \cdot log(odds) + log(1 - p)) \tag{2.6}$$

Em que L é a função *log loss*, y_i é o rótulo de cada observação, p é a probabilidade com a qual aquela observação é predita como pertencente a classe positiva e $log(odds)$ é o logaritmo da probabilidade da observação i pertencer à classe positiva. Então, o algoritmo começa com a construção do modelo base, que é obtido pela equação 2.7

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, \gamma) \tag{2.7}$$

Em que y_i é o rótulo verdadeiro da observação i e γ é o valor de $log(odds)$. Ou seja, neste passo é necessário encontrar o valor de γ que minimiza a soma do valor da função de perda para cada observação nos dados de treino. Isto é feito diferenciando a função de perda e solucionando para $\frac{dL}{d\gamma} = 0$. O valor encontrado para γ após esse passo será a primeira folha da árvore de decisão do modelo final do *gradient boosting classifier*.

Em seguida, o algoritmo repete os próximos passos M vezes, em que M é o número total de árvores que permite-se que o algoritmo adicione ao modelo.

Primeiro, calcula-se o pseudo resíduo r_{im} de cada observação. O pseudo resíduo representa a diferença entre o rótulo observado e o valor predito para aquela observação. A equação para esse passo é:

$$r_{im} = -\Big[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\Big]_{F(x)=F_{m-1}(x)} \tag{2.8}$$

Em que calcula-se o pseudo resíduo r_{im} através da derivação da função de perda com respeito à predição anterior F_{m-1} multiplicada por -1. Intuitivamente, o valor do pseudo resíduo é obtido observando o rótulo de uma observação e subtraindo o valor predito pelo última árvore do modelo até então.

Em seguida, uma árvore de regressão h_M é treinada utilizando o vetor de características de cada observação para predizer os valores dos resíduos, criando regiões terminais R_{jm} para cada folha na nova árvore. Como o número de nós na árvore é limitado por parâmetros do algoritmo, é possível que uma região terminal contenha mais de um resíduo. Então, para cada folha, calcula-se de $j = 1$ até j_M:

$$\gamma_{jm} = \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(x_i) + \gamma) \tag{2.9}$$

Em que o objetivo é encontrar o valor de γ_{jm} que minimiza a função de perda em cada folha da árvore. O somatório na equação é necessário pois é preciso agregar a perda de todos os x_i que possam pertencer em uma região terminal R_{jm}.

Finalmente, é necessário atualizar o modelo final. Esta atualização é feita segundo a equação 2.10.

$$F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{j_m} \gamma_{jm} 1(x \in R_{jm}) \tag{2.10}$$

Em que $\gamma_{jm} 1(x \in R_{jm})$ significa que utiliza-se o valor γ_{jm} se x está em uma região terminal R_{jm}. Dado que regiões terminais são exclusivas, todo x está presente em apenas uma região terminal. Este valor é multiplicado por ν, que representa a taxa de aprendizado (*learning rate*), um parâmetro do modelo que pode assumir qualquer valor entre 0 e 1, e cujo propósito é controlar o grau de contribuição que cada nova predição γ terá na predição final F_m. Finalmente, o valor é adicionado à predição anterior (F_{m-1}) para que seja realizada a predição final F_m do modelo (Junior, 2020).

2.2.6 *Extra Trees Classifier*

Este algoritmo, que também é conhecido como *Extremely Randomized Trees*, foi proposto por Geurts et al. (2006) para os problemas de classificação e regressão. Neste método, um ensemble de várias árvores de decisão é treinado com todo o *dataset* de treino, porém cada árvore recebe apenas um subconjunto aleatório das características usadas para descrever os dados, e o valor de corte para cada característica é escolhido também de forma aleatória. Ao final do modelo, uma predição é realizada agregando as predições de cada árvore e selecionando a classe que possui mais votos, esquema que recebe o nome de votação por maioria.

Para utilizar este método, três parâmetros são de extrema importância: o número M de árvores a serem criadas pelo modelo; o número K de atributos a serem escolhidos aleatoriamente e n_{min}, o número mínimo de amostras necessárias para que seja realizado o split de um nó da árvore.

Definidos esses parâmetros, o algoritmo gera M árvores que serão treinadas com todos os dados de treino, e cada árvore recebe um subconjunto aleatório de tamanho K do vetor de características que descreve os dados.

A construção de cada árvore, então, procede da seguinte maneira: a cada nó adicionado na árvore é computado um valor aleatório de corte para cada característica do subconjunto que a árvore recebeu. Para decidir qual característica será usada para aquele nó, o algoritmo avalia cada característica de acordo com um critério matemático definido como detalhe de implementação. Neste trabalho utilizou-se a impureza de Gini.

Esse procedimento é repetido até que o número de observações em uma folha seja menor ou igual que o valor definido pelo parâmetro *número mínimo de amostras em uma folha* do modelo ou até que o limite de níveis da árvore seja alcançado, caso este limite tenha sido definido.

O *Extra Trees* possui várias similaridades com o algoritmo *Random Forest*, outro *ensemble* baseado em árvores de decisão muito utilizado na área de aprendizado de máquina. Comparando os dois, espera-se que o modelo gerado pelo *Extra Trees* tenha menos viés, já que em todo nó utiliza-se todo o *dataset* de treino em vez de subconjuntos dos dados como acontece no *Random Forest*, e que a utilização de valores de corte aleatórios para cada característica aumente a variância do modelo, gerando árvores menos correlacionadas e consequentemente tornando o modelo mais robusto (Geurts et al., 2006; Désir et al., 2021).

2.2.7 Validação Cruzada

Idealmente, algoritmos de aprendizado de máquina devem ser treinados em um conjunto de dados de treino e avaliados em um conjunto de dados separado, que recebe o nome de conjunto de validação, de forma que se possa avaliar o resultado do modelo com dados que ele não tenha tido a chance de aprender ainda. No entanto, isto nem sempre é possível, pois em muitos problemas no aprendizado de máquina não se possui dados o suficiente para compor dois *datasets* independentes.

Uma forma de lidar com essa limitação é utilizando a validação cruzada por *K folds*, que está exemplificada na figura . Esse método consiste em repartir os dados de treino em K partições (*folds*) de mesmo tamanho de forma aleatória, sendo que o valor de K é decidido como detalhe de implementação. Então, para cada partição: treina-se o modelo nos outros $K - 1$ *folds* e utiliza-se a partição K_i como conjunto de validação, guardando os valores obtidos pelo modelo naquela rodada. Ao final das K iterações, o desempenho final do modelo é obtido pela média dos desempenhos em cada partição usada para validação.

Como as partições no *K fold* são geradas apenas uma vez, nenhuma observação aparece em mais de uma partição. Portanto, cada amostra no *dataset* de treino recebe a oportunidade de aparecer no conjunto de validação uma única vez e de ser treinada pelo modelo $K - 1$ vezes.

A escolha do valor de K não segue nenhuma regra, ficando a critério de implementação. Quanto maior o valor de K, espera-se que menor seja o viés do modelo, porém corre-se o risco de o modelo apresentar alta variância e tempo de treinamento exorbitantes caso escolha-se um valor muito grande (James et al., 2013). Valores como $K = 5$ ou $K = 10$ são comumente vistos na literatura, pois em testes empíricos verificou-se que esses valores geram resultados bons e que tendem a não sofrer de alto viés nem de alta variância (Kuhn e Johnson, 2013). Neste trabalho, adotou-se o valor de $K = 5$ em todos os modelos avaliados.

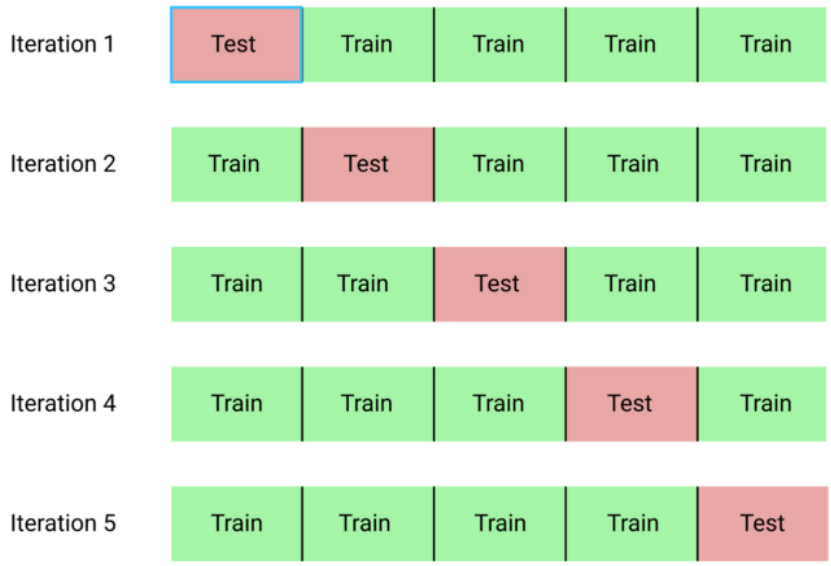

Figura 2.5: Representação visual da técnica *K fold* com 5 *folds*. A cada iteração, uma das partições é utilizada para validação enquanto as outras são usadas para treino. FONTE: Vasconcellos (2019)

3 MATERIAIS E MÉTODOS

3.1 DATASETS

Dois *datasets* foram utilizados para este trabalho, ambos com sequências de aminoácidos classificadas em duas classes: epítopo e não epítopo, sendo assim *datasets* binários. Os detalhes de cada um estão presentes na figura 3.1. A escolha por utilizar mais de um *dataset* decorreu do interesse de entender melhor o comportamento dos *ensembles* em diferentes situações.

O dataset viral, construído por Bahai et al. 2021, é composto por 4432 sequências positivas e 8460 sequências negativas. O tamanho de todas as sequências varia entre 6 a 46 aminoácidos. Para o desenvolvimento deste trabalho, necessitou-se remover da classe positiva um peptídeo que continha os caracteres "X"e "Z"em sua sequência. Esses caracteres indicam respectivamente que dois aminoácidos podem assumir aquela posição e que o aminoácido daquela posição ainda não foi identificado. Como ambos são incompatíveis com a biblioteca PyPro utilizada, pois esta não permite aplicar as técnicas de extração de características nesses casos, o *dataset* final possui um total de 12891 sequências.

O segundo dataset utilizado foi o *DeepVacPred*, disponibilizado por Yang et al. 2021 no artigo "*An in silico deep learning approach to multi-epitope vaccine design: a SARS-Cov-2 case study*". Este *dataset* é balanceado, possuindo 4925 sequências em cada classe. Além disso, todas as sequências possuem exatamente 10 aminoácidos. Pelo mesmo motivo que no *dataset* Viral, necessitou-se remover duas sequências da classe positiva e três da classe negativa, portanto o número final de peptídeos neste *dataset* é de 9845.

Dataset	Autor	Tamanho da Sequência	Classe Negativa	Classe Positiva	Tamanho Dataset
Viral	BAHAI et al. (2021)	Variado (6 a 46 aminoácidos)	8.460	4.431	12.891
DeepVacPred	YANG et al. (2021)	10 aminoácidos	4.923	4.922	9.845

Figura 3.1: Detalhes sobre os *datasets* utilizados no trabalho. FONTE: Schatzmann e Pereira (2021).

3.2 AMBIENTE DE TESTE E BIBLIOTECAS

Para a execução dos testes, utilizou-se a servidora orval do Departamento de Informática da UFPR, a qual dispõe de um processador Intel(R) Xeon(R) E5620 2.40GHz e 72GB de memória RAM. O código foi desenvolvido completamente em Python na versão 3.8.10. Optou-se por essa linguagem pela facilidade de uso e pela disponibilidade de bibliotecas relacionadas.

Os *datasets* foram manipulados utilizando a biblioteca Pandas (pandas development team, 2020). Para o cálculo da extração das características foram utilizados trechos da implementação de Schatzmann e Pereira (2021) e o módulo PyPro da biblioteca PyDPI, o qual disponibiliza

ferramentas para computar características estruturais e físico-químicas de peptídeos a partir de sequências de aminoácidos (Cao et al., 2013). Para o treino e validação dos *ensembles*, utilizou-se a biblioteca Scikit-learn 1.0.1 (Pedregosa et al., 2011), a qual implementa os algoritmos utilizados e a função *GridSearchCV*, utilizada para fazer a busca por hiper parâmetros de cada modelo.

3.3 CARACTERÍSTICAS UTILIZADAS PARA REPRESENTAR OS AMINOÁCIDOS

Todas as características utilizadas nesse trabalho foram obtidas utilizando o pacote PyPro, com exceção das características AAP e AAT, as quais foram extraídas utilizando o código fornecido por Schatzmann e Pereira (2021). Em ambos os *datasets*, o vetor de característica final usado para representar cada peptídeo é obtido pela concatenação das características apresentadas nesta seção.

3.3.1 Composição de aminoácidos

A composição de aminoácidos é composta por um vetor de 20 posições, uma para cada aminoácido, e calculado utilizando a fórmula 3.1:

$$AAC(P) = (f_1, f_2, f_3, \ldots, f_{20}) \tag{3.1}$$

Em que $f_i = \frac{R_i}{N}$ é a porcentagem da composição da sequência que é do tipo do aminoácido indexado por i, R_i é a quantidade de aminoácidos do tipo i que aparecem na sequência e N é o tamanho do peptídeo.

3.3.2 Escala de antigenicidade em pares

A escala de antigenicidade em pares (AAP) mede a frequência com a qual um par de aminoácidos ocorre em cada classe no dataset usado para treinamento do modelo (Chen et al., 2012). Ela é calculada em duas etapas: primeiramente gera-se um dicionário de escala de antigenicidade para todos os pares possíveis de aminoácidos (totalizando 400 combinações), em que o valor de cada chave é calculado da seguinte forma:

$$Raap_i = \log \frac{f^+aap_i}{f^-aap_i} \tag{3.2}$$

Sendo que i é o par de aminoácidos, f^+aap_i é a frequência de ocorrência daquele par na classe positiva e f^-aap_i é a frequência de ocorrência da classe negativa. Ao final do cálculo de todos os pares, os valores são normalizados com a seguinte fórmula 3.3:

$$2 * \frac{Raap_i - \min Raap}{\max Raap - \min Raap} - 1 \tag{3.3}$$

Finalmente, o valor final da escala é obtido calculando a média aritmética dos 400 possíveis valores para cada peptídeo, como pode ser visto na fórmula 3.4:

$$AAP = \frac{\sum_1^{n-1} Raap_i}{n-1} \tag{3.4}$$

Na qual n é o tamanho do peptídeo e i é cada um dos possíveis pares de aminoácidos. Ao final desse cálculo, o valor final representa a relação dos aminoácidos em pares em cada um dos peptídeos.

3.3.3 Escala de antigenicidade em trio

A escala de antigenicidade em trios (AAT) é calculada de maneira muito semelhante à característica AAP, porém considerando trios em vez de pares de aminoácidos, elevando o espaço amostral de possibilidades para 8000 possíveis combinações (Yao et al., 2012).

As etapas de geração do dicionário e de normalização ocorrem de forma idêntica ao AAP, apenas considerando os trios de aminoácidos em vez dos pares. Similarmente, o cálculo para obter o valor final dessa característica difere levemente do cálculo da AAP, sendo feito de acordo com a fórmula 3.5:

$$AAT = \frac{\sum_1^{n-2} Raap_i}{n-2} \tag{3.5}$$

Na qual itera-se até $n-2$ pois é onde encontramos a última tripla do peptídeo. Este cálculo é feito para cada peptídeo, e ao final desse processo a característica AAT representa a escala de antigenicidade de cada tripla de aminoácidos de cada peptídeo.

3.3.4 Composition Transition Distribution

O conceito de *composition transition distribution* (CTD) foi utilizado com sucesso em vários problemas de classificação baseados em sequências de aminoácidos (Manavalan et al., 2018). Essa característica consiste em descritores de 7 propriedades dos aminoácidos: hidrofobicidade, volume normalizado de Van der Waals, polaridade, polarizabilidade, carga, estrutura secundária e acessibilidade do solvente (Wang et al., 2017). De acordo com essas propriedades, os 20 aminoácidos são divididos em três grupos: polar, neutro e hidrofóbico. Cada aminoácido é codificado por um índice de acordo com o grupo ao qual ele pertence. Finalmente, cada sequência de aminoácidos é transformada em um vetor numérico para o cálculo das propriedades CTD.

A Composição (C) consiste no número de aminoácidos de cada um dos grupos dividido pelo número total de aminoácidos de um peptídeo. A Transição (T) é a frequência em porcentagem com a qual aminoácidos de um grupo são seguidos por um aminoácido de outro grupo na sequência do peptídeo avaliado. A Distribuição (D) consiste de cinco valores para cada um dos grupos, medindo o comprimento da sequência do peptídeo com a qual o primeiro, 25%, 50%, 75% e 100% dos aminoácidos de cada grupo estão localizados, respectivamente. Para cada uma das

7 propriedades, o CTD calcula 21 características, e portanto o tamanho final do vetor dessa característica possui 147 posições (Dubchak et al., 1995).

3.4 METODOLOGIA

3.4.1 Busca por hiper parâmetros de cada modelo

Para o desenvolvimento do trabalho, foram utilizadas as implementações *AdaBoostClassifier*, *GradientBoostingClassifier* e *ExtraTreesClassifier* da biblioteca *Scikit-learn*. No caso do método *gradient boosting*, utilizou-se também a implementação *XGBClassifier* (Chen e Guestrin, 2016) para comparar a performance com o modelo clássico. O treinamento de cada modelo individual foi realizado utilizando o método *GridSearchCv*, também do *Scikit-learn*. Esse método tem como objetivo gerar uma grade com todas as combinações de valores de hiper parâmetros investigados, treinar o modelo com cada combinação utilizando validação cruzada e retornar o conjunto de hiper parâmetros que maximiza a métrica de avaliação definida pelo parâmetro *scoring* do *GridSearchCV*. Para todos os testes, utilizou-se 5 *folds* de validação. Os valores investigados de cada hiper parâmetro podem ser vistos na tabela 3.1.

3.4.2 Métricas de Avaliação

Visto que este trabalho lida com a classificação de epítopos, considerou-se para os testes que os epítopos preditos de forma correta compõem o conjunto de verdadeiros positivos (TP, do inglês *true positive*), os peptídeos não-epitópicos corretamente preditos compõem os verdadeiros negativos (TN, do inglês *true negative*), e os epítopos erroneamente classificados e peptídeos incorretamente preditos como epítopos formam os conjuntos de falso negativo (FN) e falso positivo (FP), respectivamente.

Para avaliação de cada modelo utilizaram-se métricas de avaliação comuns para problemas de classificação (Bahai et al., 2021), sendo estas: acurácia, precisão, *recall*, F1 Score, coeficiente de correlação de Matthews (MCC) e área sob a curva característica de operação do receptor (ROC_{AUC}). A escolha por esse conjunto de métricas se deu pelo interesse em analisar o comportamento de cada modelo sob diferentes perspectivas. Em última instância, definiu-se que o melhor modelo retornado pelo *GridSearchCV* seria aquele que maximizasse o valor de MCC, isso pois durante a revisão de literatura, pode-se ver que esta é melhor métrica para medir o desempenho de classificadores binários, sendo especialmente preferida em problemas com dados desbalanceados, como frequentemente é o caso em problemas de classificação baseados em peptídeo (Galanis et al., 2021; Manavalan et al., 2018; Bahai et al., 2021; Raschka, 2014). A fórmula para o cálculo dessa característica é:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{3.6}$$

Tabela 3.1: Hiper parâmetros otimizados pelo *GridSearchCV*

Modelo	Parâmetros	Valores Investigados
Adaboost	Número de estimadores	5, 10, 25, 50, 100, 200, 300, 400, 500, 750, 1000
	Algoritmo	'SAMME', 'SAMME.R'
	Taxa de aprendizado	1, 0.5, 0.25, 0.1, 0.05, 0.01
Gradient Boosting	Número de estimadores	5, 10, 25, 50, 100, 200, 300, 400, 500, 750, 1000
	Número máximo de features considerada por split	"sqrt", None
	Taxa de aprendizado	1, 0.5, 0.25, 0.1, 0.05, 0.01
	Profundidade Máxima da Árvore	3, 5, 8, 16
XGBoost	Número de estimadores	5, 10, 25, 50, 100, 200, 300, 400, 500, 750, 1000
	Taxa de aprendizado	1, 0.5, 0.25, 0.1, 0.05, 0.01
	Profundidade Máxima da Árvore	3, 5, 8, 16
	Peso mínimo do nó filho	1, 3, 5
	Subamostra	0.6, 0.7, 0.8, 0.9
	Subamostra de colunas por árvore	0.6, 0.7, 0.8, 0.9
	Gama	0.0, 0.01, 0.05, 0.1, 0.25, 0.5, 1
	Regularização L1	0, 0.01, 0.05, 0.1, 0.25, 0.5, 1
ExtraTrees	Número de estimadores	5, 10, 25, 50, 100, 200, 300, 400, 500, 750, 1000
	Número máximo de features considerada por split	"sqrt", "log2", None
	Número mínimo de amostras necessárias para dividir um nó interno	2, 8, 16, 32, 64, 128
	Profundidade Máxima da Árvore	2, 8, 16, 32

Diferente de todas as outras métricas, o valor do MCC está definido no intervalo $[-1, +1]$, em que o valor máximo representa que o modelo acertou todas as predições, enquanto que o valor mínimo significa que nenhuma predição foi correta. Caso o MCC seja 0, interpreta-se que a predição do modelo é equivalente à uma predição aleatória (Galanis et al., 2021).

Como o *GridSearchCV* foi utilizado com validação cruzada de 5 *folds*, o score final em cada métrica para cada modelo é obtido pelo cálculo da média dos valores de cada métrica em cada um dos *folds*.

4 RESULTADOS

Nesta seção são apresentados os resultados obtidos na etapa de otimização de hiper parâmetros para ambos os *datasets*. Os valores para cada algoritmo representam o score obtido pela melhor combinação para cada algoritmo. No caso de empate, considerou-se que o melhor modelo é aquele que apresenta o menor tempo de treino.

Para cada *dataset* são apresentados os valores investigados por parâmetro e o comportamento de cada parâmetro com respeito ao score na métrica MCC. Ao final de cada subseção são apresentados os valores de todas as métricas utilizadas para cada um dos métodos.

4.1 DATASET VIRAL

O *dataset* Viral de Bahai et al. (2021) foi o principal objeto de estudo neste trabalho, uma vez que por ser desbalanceado e com sequências de tamanho variável, ele é mais representativo dos tipos de dados que se lida no problema de classificação de epítopos.

4.1.1 Adaboost

Para este método, foram investigados os parâmetros número de estimadores (*n_estimators*), algoritmo (*algorithm*) e taxa de aprendizado (*learning_rate*). O número de estimadores consiste no número total de árvores de decisão que é permitido ser adicionado ao modelo. Valores maiores permitem que o modelo aprenda padrões profundos nos dados, porém também podem levar ao *overfitting*. O parâmetro algoritmo determina o algoritmo que será utilizado para determinar o poder de voto de cada árvore na fórmula 2.3. No esquema *SAMME*, cada classificador fraco utiliza um valor de α diferente, enquanto o esquema *SAMME.R* atribui um peso igual para todos os classificadores e trabalha com a probabilidade de uma observação pertencer a cada classe, em vez de cada classificador fraco fazer uma predição binária. De acordo com a documentação, "o algoritmo *SAMME.R* costuma convergir mais rápido do que o *SAMME*, alcançando um valor menor de erro de teste em menos iterações de *boosting*"(Pedregosa et al., 2011). Por fim, o parâmetro taxa de aprendizado se refere ao valor de v em 2.3. No total, 132 combinações foram testadas. Os resultados em termos de MCC da variação de cada parâmetro podem ser vistos na figura 4.1.

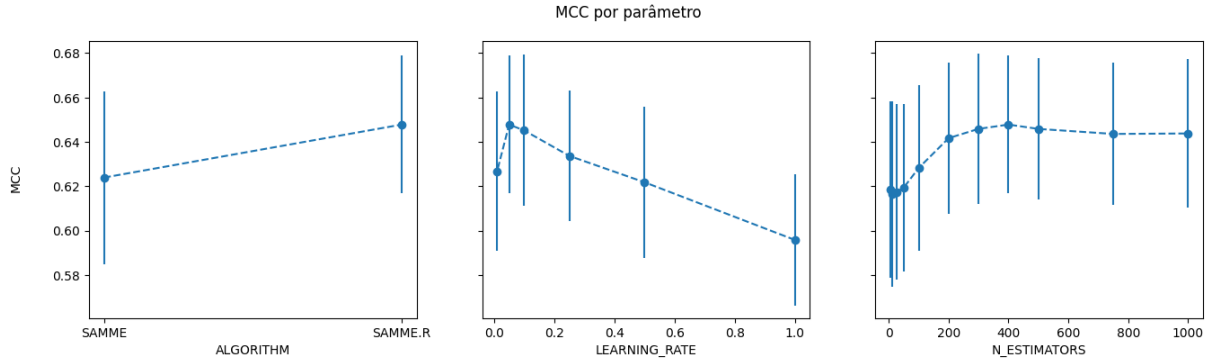

Figura 4.1: Score de MCC obtido com a variação de cada parâmetro investigado no algoritmo Adaboost para o *dataset* viral. Os pontos representam a média do valor nos 5 *folds*, enquanto as linhas verticais representam a variância do MCC obtida dentre os 5 *folds*. FONTE: O autor (2022).

Para a geração de cada gráfico, fixaram-se o valor de todos os parâmetros no valor ótimo encontrado e observou-se o resultado da variação apenas de um parâmetro por vez. Como pode-se ver, os resultados corroboram a afirmação dos autores da biblioteca, pois o algoritmo *SAMME.R* apresentou resultados melhores do que o *SAMME*. O melhor valor de número de árvores encontrado foi de 400 estimadores, a partir do qual a adição de novos classificadores piora os resultados. A taxa de aprendizado apresenta melhoria entre 0.01 e 0.05, este que foi o valor ótimo encontrado, e assim como no número de estimadores, ao passar esse valor o modelo apresenta resultados piores.

4.1.2 *Gradient Boosting*

Para o modelo clássico do*GradientBoostingClassifier*, consideraram-se os parâmetros número de estimadores, número máximo de características (*max_features*) a serem consideradas a cada split (*max_features*), a taxa de aprendizado e a profundidade máxima permitida para cada árvore de classificação (*max_depth*). Com o parâmetro *max_features* pode-se controlar o tempo de aprendizado do modelo, reduzindo o número de características a serem consideradas por split, embora isso possa reduzir a acurácia do modelo em certos casos. Com o parâmetro *max_depth*, controla-se diretamente a complexidade do modelo, uma vez que se forem permitidas árvores com profundidades grandes, cada classificador fraco se torna mais capaz de aprender padrões sutis nos dados, porém corre-se o risco de *overfitting*, além de aumentar em muito o tempo de treinamento do modelo. No total, foram testadas 528 combinações de parâmetros. Os resultados estão presentes na figura 4.2.

O mesmo comportamento visto no *Adaboost* pode ser visto também nesse modelo para os parâmetros *learning_rate* e *n_estimators*, com ambos apresentando queda de performance após os valores ótimos, 0.1 e 50, respectivamente. A melhor profundidade máxima foi obtida com o valor *default* de 3 níveis, com o pior resultado sendo obtido quando a profundidade foi a maior permitida, 16. Quanto ao número máximo de características, verifica-se que permitir que todas as características sejam avaliadas em todo split provê os melhores resultados.

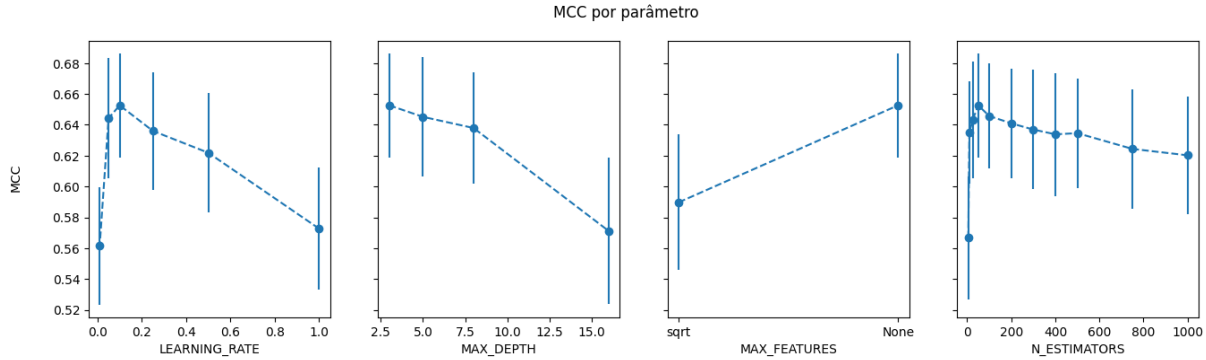

Figura 4.2: Score de MCC obtido com a variação de cada parâmetro investigado no algoritmo *GradientBoosting-Classifier* para o *dataset* viral. FONTE: O autor (2022).

4.1.3 *XGBClassifier*

O eXtreme Gradient Boosting é uma implementação otimizada e distribuída do algoritmo de *Gradient Boosting*, feita para facilitar o treinamento deste algoritmo, que tende a ter tempos longos de treino. Os parâmetros que essa implementação fornece para customizar o comportamento do algoritmo são vários, e portanto necessitou-se fazer os testes em duas rodadas. Na primeira, investigaram-se os parâmetros número de estimadores, taxa de aprendizado, profundidade máxima e peso mínimo do nó filho . O peso mínimo do nó filho se refere à soma mínima de peso de instância necessário para se criar um nó em um split, e pode ser interpretado como o número mínimo de amostras necessárias para que seja considerado um novo split. Os resultados da primeira rodada estão na figura 4.3. No total, considerou-se 792 combinações para essa rodada.

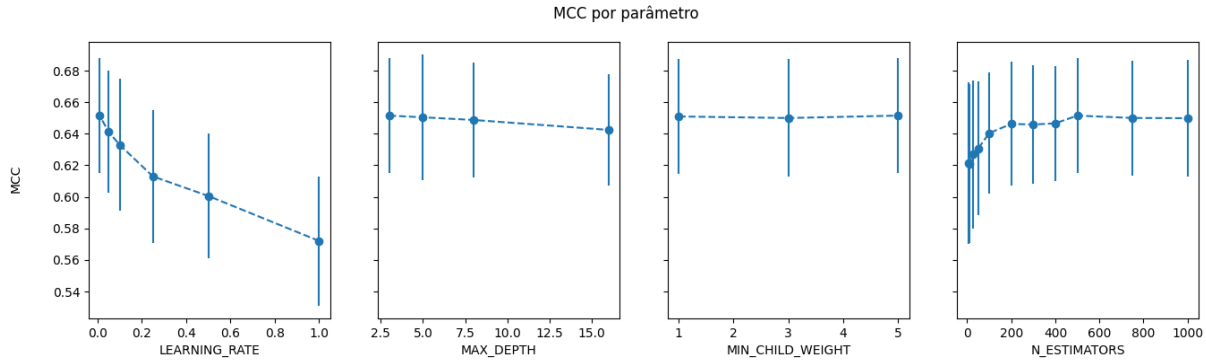

Figura 4.3: Score de MCC obtido na primeira rodada de variação de cada parâmetro investigado no algoritmo *XGBClassifier* para o *dataset* viral. FONTE: O autor (2022).

A taxa de aprendizado ótima para o *XGBClassifier* neste *dataset* foi de 0.01, com o resultado piorando toda vez que esse valor foi aumentado. A melhor profundidade máxima foi de 3 níveis, e o melhor valor para o min_child_weight foi de 5, embora para esse parâmetro verificou-se que os valores investigados apresentaram resultados muito similares. Por fim, o número ótimo de estimadores foi de 500, a partir do qual os resultados pioraram levemente.

Na segunda rodada os valores ótimos encontrados na primeira foram configurados e variou-se um novo conjunto de parâmetros, que são: a razão de subamostragem das colunas por árvore (*colsample_bytree*), o valor de gama (*gamma*), o fator de regularização α (*reg_alpha*) e a subamostragem (*subsample*). Colunas no parâmetro col_sample_bytree se refere às colunas do vetor de característica, pois esse parâmetro é usado para limitar o número de características investigadas a cada split. Gama representa o valor mínimo de redução da perda necessária para que se considere um novo split em uma folha da árvore, e quanto maior seu valor mais conservador será o classificador. O parâmetro reg_alpha é o fator de regularização L1 aplicado nos pesos das folhas, e similarmente ao parâmetro gamma, quanto maior o valor de reg_alpha, mais conservador é o algoritmo. Por fim, o parâmetro subamostragem controla a quantidade do *dataset* de treino que é amostrada a cada iteração do processo de *boosting*. No total, foram testadas 784 novas combinações nesta rodada, cujos resultados podem ser vistos na figura 4.4.

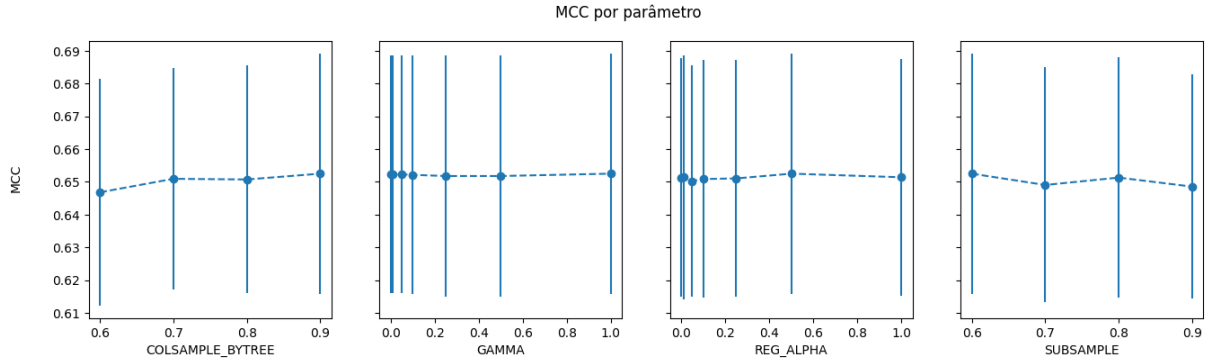

Figura 4.4: Score de MCC obtido na segunda rodada de variação de cada parâmetro investigado no algoritmo *XGBClassifier* para o *dataset* viral. Os valores de learning_rate, max_depth, min_child_weight e n_estimators foram configurados nos valores ótimos encontrados na rodada anterior. FONTE: O autor (2022).

Foi possível observar que a performance do algoritmo melhora conforme mais características são permitidas serem consideradas a cada split, pois o valor ótimo de colsample_bytree foi de 0.9. Curiosamente, verificou-se que o valor de gamma não representou diferença significativa quando foi variado, porém o melhor resultado foi obtido quando esse parâmetro assumiu o valor 1. Para reg_alpha, o melhor valor encontrado foi de 0.5, e finalmente a subamostragem apresentou melhores resultados em seu menor valor testado, 0.6. No geral, os parâmetros testados nessa rodada não fizeram tanta diferença quanto na rodada anterior, o que era de se esperar considerando a documentação deste método. Ainda assim, foi possível melhorar levemente o score em termos de MCC em comparação com o score obtido na primeira rodada.

4.1.4 *ExtraTrees*

No modelo *ExtraTrees*, variou-se o número de estimadores, o número máximo de características a serem consideradas a cada split, o número mínimo de amostras necessárias para se considerar um split (*min_samples_split*) e a profundidade máxima permitida para cada árvore do modelo. Novos splits nas árvores de decisão só são gerados se o número de amostras

separadas pelo split for maior ou igual que o parâmetro *min_samples_split*. Se for utilizado um valor pequeno para este parâmetro, espera-se que a correlação entre as árvores seja menor pois cada árvore pode gerar árvores maiores e mais especializadas. No total, testaram-se 792 combinações para este modelo, cujos resultados estão na figura 4.5.

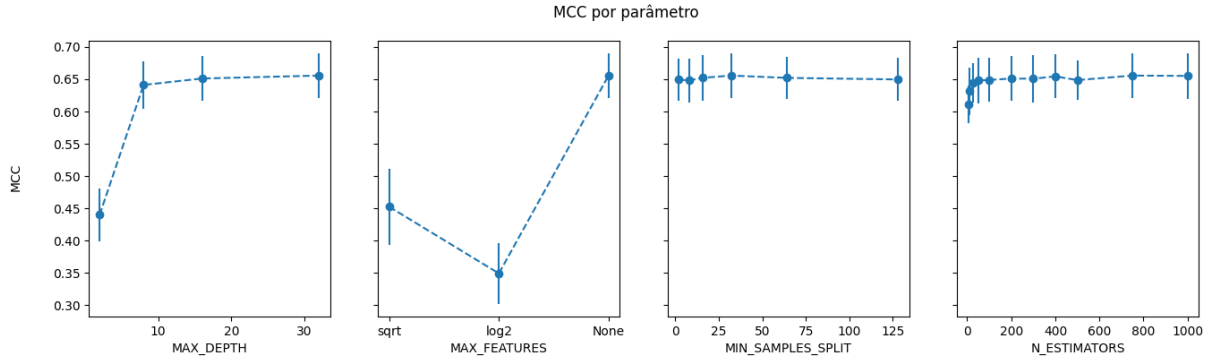

Figura 4.5: Score de MCC obtido com a variação de cada parâmetro investigado no algoritmo *ExtraTreesClassifier* para o *dataset* viral. FONTE: O autor (2022).

O melhor número de estimadores para esse modelo foi de 750 árvores, embora pode-se ver que a performance a partir de 100 árvores se mantém relativamente igual. O melhor número mínimo de amostras para se considerar um split encontrado foi de 32 amostras. Novamente, o melhor valor para o parâmetro *max_features* foi considerando todas as características, com o pior resultado sendo obtido na configuração log2, o que era de se esperar pois nessa configuração apenas 7 características são consideradas por split. Para o *dataset* viral, observou-se que árvores mais profundas possuíram maior poder de classificação do que árvores rasas, o que indica que os padrões nesse *dataset* requerem modelos mais sofisticados para aprender.

4.1.5 Performance de todos os modelos

Na figura 4.6 estão presentes as performances de cada classificador nas métricas avaliadas. Para a avaliação das métricas precisão, *recall* e F1 Score deu-se mais importância para os valores referentes à classe positiva, pois essa é de maior importância do que a predição correta da classe negativa.

	Adaboost	Gradient Boosting	eXtreme Gradient Boosting	Extra Trees
ROC AUC	0.909637	0.908916	0.911314	0.908934
Acurácia	0.842372	0.844312	0.844234	0.845785
Precisão (classe positiva	0.797660	0.797246	0.797690	0.806564
Recall (classe positiva)	0.733463	0.741366	0.740688	0.733693
F1 score (classe positiva)	0.762417	0.766653	0.766508	0.766565
Precisão (classe negativa)	0.865881	0.869121	0.868811	0.866588
Recall (classe negativa)	0.899409	0.898227	0.898463	0.904492
F1 score (classe negativa)	0.881873	0.883017	0.882960	0.884663
precision_micro	0.842372	0.844312	0.844234	0.845785
precision_macro	0.831770	0.833183	0.833250	0.836576
MCC	0.647807	0.652645	0.652495	0.655216

Figura 4.6: Performance de cada modelo em todas as métricas avaliadas para o *dataset* viral. Células com cor verde representam o melhor score obtido dentre todos os modelos para aquela métrica, enquanto células com cor vermelha representam os piores resultados para aquela métrica. FONTE: O autor (2022).

Como pode-se ver, para o *dataset* viral o modelo *Adaboost* apresentou resultados piores que todos os outros modelos em quase todas as métricas. O modelo *ExtraTrees* apresentou o melhor MCC, acurácia e precisão, porém ficou ligeiramente atrás do modelo *Gradient Boosting* nas métricas *recall* e F1 score. O modelo *eXtreme Gradient Boosting* se destaca apenas na métrica ROC AUC.

4.2 DATASET DEEPVACPRED

Dado que o *dataset* DeepVacPred foi incluído no escopo deste trabalho apenas como ferramenta de comparação para verificar se existem diferenças contrastantes na performance dos modelos em um cenário balanceado e com sequências de tamanho iguais, e dado que os parâmetros investigados para os modelos foram os mesmos, essa seção apresenta os resultados neste *dataset* de forma mais resumida.

4.2.1 Adaboost

A diferença mais notável deste método neste *dataset* pode ser vista no parâmetro n_estimators, em que o valor ótimo foi de 750 classificadores e nota-se que há uma diferença maior entre os resultados obtidos com poucos e com muitos classificadores quando comparado ao *dataset* viral. Quanto aos outros dois parâmetros, o comportamento foi o mesmo, com os melhores valores sendo o algoritmo *SAMME.R* e a taxa de aprendizado 0.05.

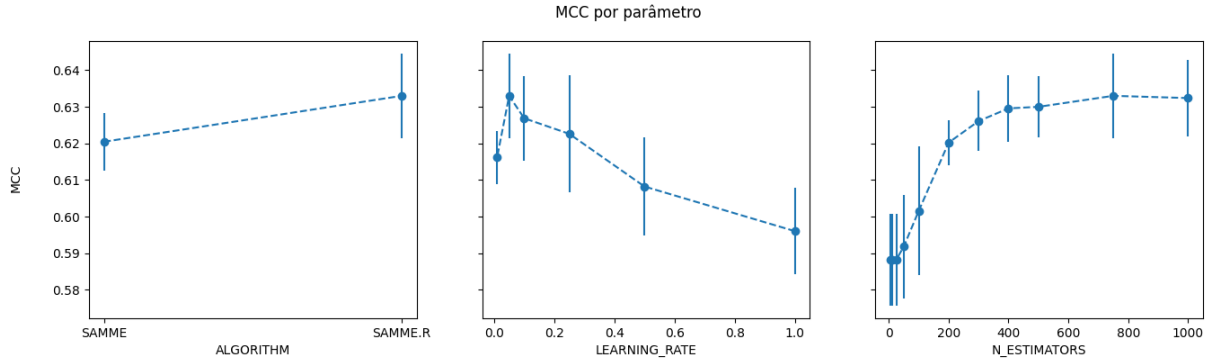

Figura 4.7: Score de MCC obtido com a variação de cada parâmetro investigado no algoritmo Adaboost para o *dataset* DeepVacPred. FONTE: O autor (2022).

4.2.2 *Gradient Boosting Classifier*

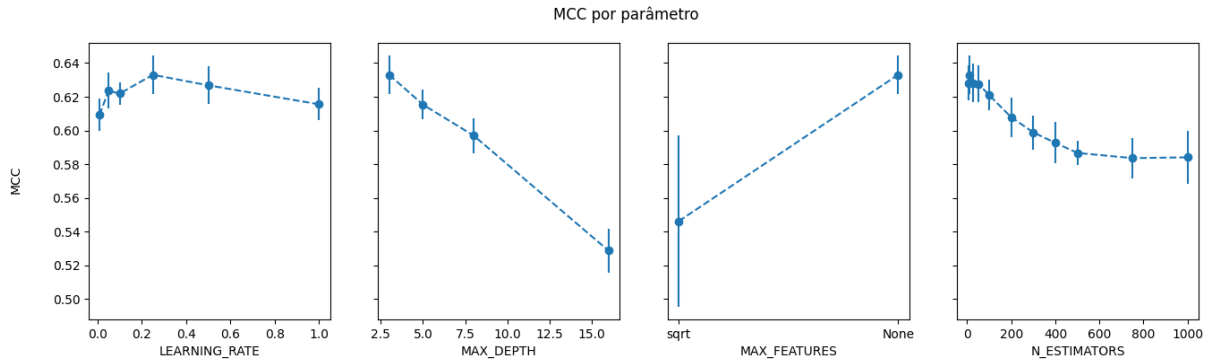

Figura 4.8: Score de MCC obtido com a variação de cada parâmetro investigado no algoritmo *Gradient Boosting Classifier* para o *dataset* DeepVacPred. FONTE: O autor (2022).

No *Gradient Boosting Classifier* o comportamento difere um pouco entre os *datasets*. O número de estimadores ótimo para o DeepVacPred foi de apenas 10 classificadores, com uma taxa de aprendizado um pouco maior, 0.25, que no *dataset* viral. Quanto aos outros dois parâmetros, o comportamento foi o mesmo para os dois *datasets*. Chama atenção, no entanto, que a variância entre os 5 *folds* é muito menor para o DeepVacPred, o que mostra que, para esse método, este *dataset* é consideravelmente mais simples de aprender do que o viral.

4.2.3 *XGBClassifier*

Comparado ao *dataset* viral, houve bastante diferença no comportamento deste algoritmo. A taxa de aprendizado melhora até chegar no valor ótimo, 0.5, a partir do qual os resultados pioram. A variação da profundidade máxima teve um efeito mais drástico, sendo o melhor valor encontrado na configuração mínima de 3 níveis. Similarmente, o número de estimadores ótimo foi 10, com uma drástica queda de performance quando são adicionados mais classificadores ao modelo. Quanto ao parâmetro min_child_weight, neste *dataset* encontrou-se o valor ótimo de 5.

Da mesma forma que no *dataset* viral, realizaram-se os testes em duas rodadas. Os resultados da segunda rodada estão na figura 4.10:

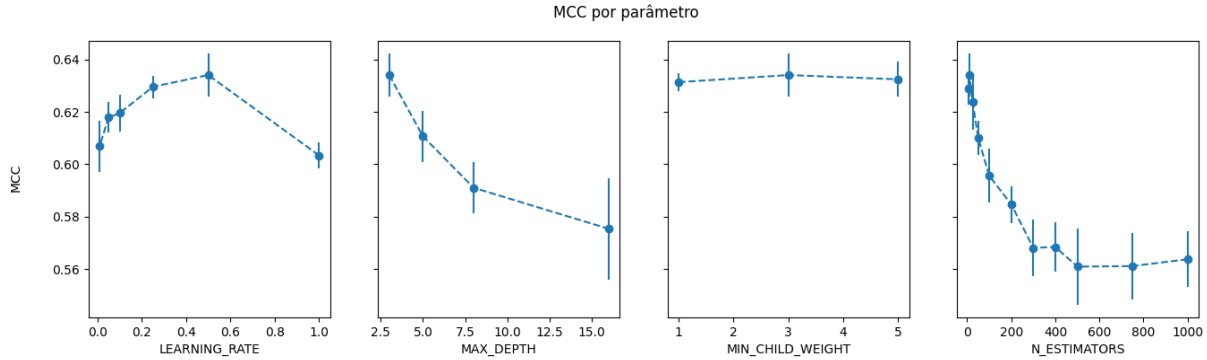

Figura 4.9: Score de MCC obtido na primeira rodada de variação de cada parâmetro investigado no algoritmo *XGBClassifier* para o *dataset* DeepVacPred. FONTE: O autor (2022).

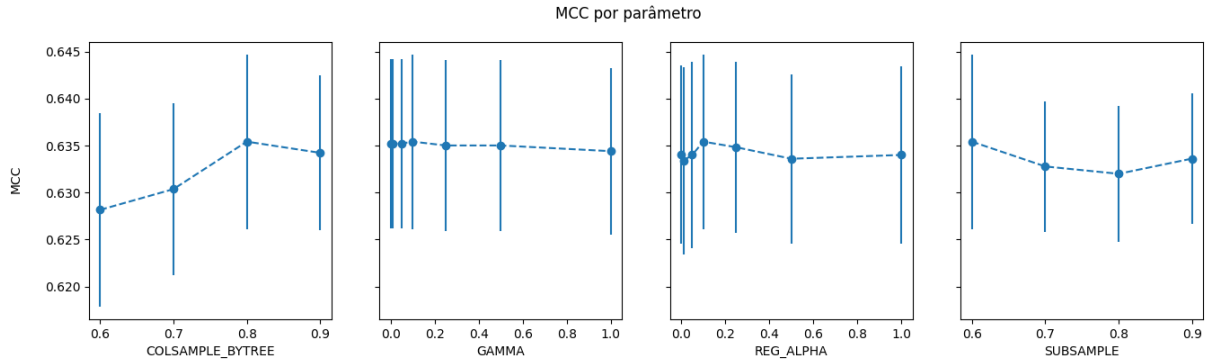

Figura 4.10: Score de MCC obtido na segunda rodada de variação de cada parâmetro investigado no algoritmo *XGBClassifier* para o *dataset* DeepVacPred. Os valores de learning_rate, max_depth, min_child_weight e n_estimators foram configurados nos valores ótimos encontrados na rodada anterior. FONTE: O autor (2022).

No DeepVacPred, a variação dos parâmetros na segunda rodada também não fez muita diferença, porém similarmente melhorou levemente os resultados do algoritmo. Os valores ótimos encontrados foram de 0.8 para colsample_bytree, 0.1 para gamma e reg_alpha, e finalmente a subamostragem teve o mesmo valor ótimo que no *dataset* viral, com 0.6.

4.2.4 *Extra Trees Classifier*

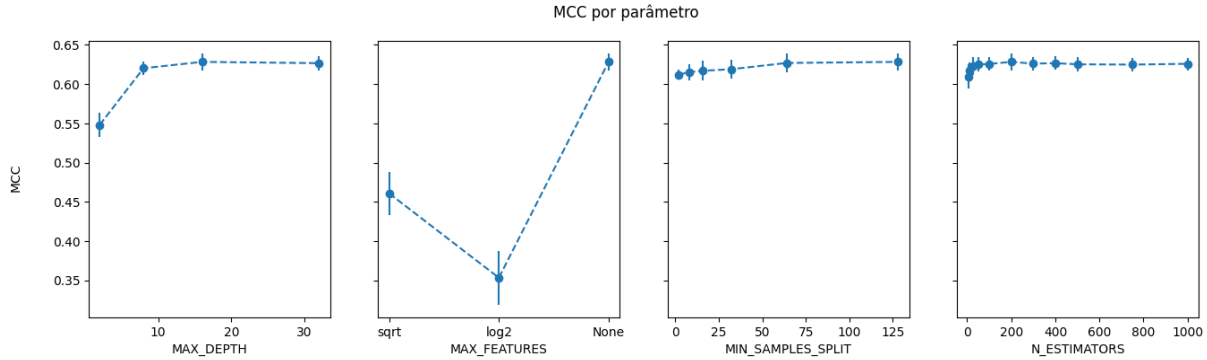

Figura 4.11: Score de MCC obtido com a variação de cada parâmetro investigado no algoritmo *Extra Trees* para o *dataset* DeepVacPred. FONTE: O autor (2022).

Para o *Extra Trees Classifier*, a diferença que mais chama atenção está no parâmetro n_estimators, em que neste *dataset* o valor ótimo é 200 classificadores. Os *datasets* diferem também nos melhores valores de max_depth e min_samples_split, sendo respectivamente 16 e 128 os valores ideais para estes parâmetros no DeepVacPred

4.2.5 Performance de todos os modelos

Assim como para o *dataset* viral, aqui são ilustrados os resultados de todos os classificadores nas métricas avaliadas.

	Adaboost	Gradient Boosting	eXtreme Gradient Boosting	Extra Trees
ROC AUC	0.898058	0.895713	0.897782	0.898106
Acurácia	0.816455	0.816455	0.816963	0.814119
Precisão (classe positiva	0.817172	0.815831	0.813839	0.815962
Recall (classe positiva)	0.815563	0.817596	0.822062	0.811299
F1 score (classe positiva)	0.816316	0.816633	0.817883	0.813569
Precisão (classe negativa)	0.815899	0.817323	0.820302	0.812463
Recall (classe negativa)	0.817350	0.815319	0.811865	0.816944
F1 score (classe negativa)	0.816572	0.816242	0.816012	0.814642
precision_micro	0.816455	0.816455	0.816963	0.814119
precision_macro	0.816535	0.816577	0.817071	0.814213
MCC	0.632992	0.633034	0.634034	0.628334

Figura 4.12: Performance de cada modelo em todas as métricas avaliadas para o *dataset* DeepVacPred. Células com cor verde representam o melhor score obtido dentre todos os modelos para aquela métrica, enquanto células com cor vermelha representam os piores resultados para aquela métrica. FONTE: O autor (2022).

Vê-se que DeepVacPred, pelas peculiaridades de sua construção, apresenta o pior resultado quando aprendido utilizando o método *ExtraTreesClassifier*, diferente do comportamento visto no *dataset* viral. Isto pode ser facilmente explicado, uma vez que os dados no DeepVacPred são muito mais uniformes e portanto a aleatoriedade do método *ExtraTrees* não é tão interessante neste caso. Com exceção das métricas ROC AUC e precisão, o melhor classificador para este *dataset* foi o *eXtreme Gradient Boosting*.

5 CONSIDERAÇÕES FINAIS

Neste trabalho buscou-se entender os obstáculos que impedem que a produção de vacinas contra patógenos altamente infecciosos ocorra com a velocidade necessária para impedir que catástrofes como a pandemia de COVID-19 se repitam. Identificou-se que métodos tradicionais de design de vacinas requerem muito tempo de desenvolvimento, e que uma alternativa mais ágil e com menos efeitos adversos é o design de vacinas baseadas em epítopos, regiões de um antígeno à qual linfócitos se ligam. Após uma revisão de literatura, verificou-se que uma etapa importante para o design de vacinas por este método é a predição de epítopos lineares de células B, e que há poucas abordagens por *ensemble* para a realização dessa etapa. Neste sentido, propôs-se investigar a performance de quatro modelos de *ensemble* que implementam três algoritmos diferentes para a tarefa de classificação binária de epítopos lineares de células B, a partir do treino e validação cruzada em dois *datasets* de epítopos virais.

Esta seção visa, então, apresentar as considerações finais sobre as oportunidades e limitações da abordagem estudada identificadas com base nos resultados obtidos, bem como trazer algumas possíveis sugestões de trabalhos futuros na área.

5.1 APLICABILIDADE DOS MÉTODOS

O rigor científico determina que métodos de aprendizado de máquina sejam avaliados não apenas com o *dataset* de treino, mas também utilizando um grupo de dados independente, os quais o algoritmo não tenha observado durante o treino, i.e., um *dataset* de teste. No entanto, devido às complexidades da identificação de epítopos *in-vitro* as quais foram citadas ao longo do texto, verificou-se que há uma baixa disponibilidade de *datasets* de qualidade para o problema de predição de epítopos de células B. Além disso, como pôde-se observar comparando a performance dos modelos nos dois *datasets*, a generalização da performance dos algoritmos é dificultada pois ela está condicionada, em parte, pelas peculiaridades da forma como os dados de treino foram construídos.

Ainda assim, a validação cruzada é um procedimento padrão na área de aprendizado de máquina para contornar os obstáculos citados. Com ela, obteve-se valores relevantes de MCC e de *recall*, que são as métricas mais importantes para o problema em questão, uma vez que a predição correta de sequências demonstradamente epitópicas é mais importante do que a classificação correta de sequências não epitópicas. Para o *dataset* viral, que é mais representativo dos dados inerentes ao problema de classificação de epítopos, o *ensemble ExtraTreesClassifier* foi o melhor modelo, com um MCC de 0.655. Já para o *dataset* DeepVacPred, a implementação *eXtreme Gradient Boosting* apresentou os melhores resultados, com um MCC de 0.634. Portanto, conclui-se que os resultados apresentados neste trabalho são interessantes e indicam que a utilização de *ensembles* pode ser um caminho viável para a predição de epítopos lineares de

células B, desde que sejam levados em consideração as características dos dados usados para treino dos modelos e as limitações de cada método de aprendizado. É de suma importância, também, relembrar que a predição de epítopos é apenas um dos passos no design de vacinas baseadas em epítopos, e que epítopos preditos por modelos computacionais, independente de quão confiáveis sejam estes modelos, necessitam de validação *in-vitro* e *in-vivo* antes de serem utilizados para a confecção de vacinas.

5.2 TRABALHOS FUTUROS

Verificou-se que há uma grande necessidade da criação de *datasets* experimentalmente validados e que forneçam um volume suficientemente grande de dados para o treinamento dos modelos de *ensemble*, visto que estes são algoritmos de aprendizado supervisionado e portanto suas performances estão condicionadas à disponibilidade de dados de qualidade para o aprendizado. Também, foi possível identificar a necessidade de se investigar novas formas de caracterizar os dados, uma vez que não encontrou-se na literatura um conjunto consenso de características para a discriminação de sequências de aminoácidos em epítopos e não-epítopos, o que dificulta a comparação entre trabalhos dessa área. Além disso, entende-se como interessante para a continuação desse trabalho a avaliação da performance de um *framework* que incorpore os diferentes algoritmos estudados para a predição conjunta de epítopos de acordo com a técnica de generalização empilhada (*stacking*) descrita na seção 2.2.2 deste texto.

REFERÊNCIAS

Abbas, A. K., Lichtman, A. H. e Pillai, S. (2021). *Imunologia Básica: Funções e distúrbios do sistema imunológico.* GEN Guanabara Koogan.

Baby, D., Devaraj, S. J., Hemanth, J. e M., A. R. (2021). Leukocyte classification based on feature selection using extra trees classifier: atransfer learning approach,. *Turkish Journal of Electrical Engineering and Computer Sciences*, 29(8). Disponível em https://doi.org/10.3906/elk-2104-183 Acessado em 31/08/2022.

Bahai, A., Asgari, E., Mofrad, M. R. K., Kloetgen, A. e McHardy, A. C. (2021). Epitope-vec: linear epitope prediction using deep protein sequence embeddings. *Bioinformatics*, 37(23):4517–4525. Advance online publication. Disponível em https://doi.org/10.1093/bioinformatics/btab467 Acessado em 31/08/2022.

Breiman, L. (1998). Arcing the edge. Relatório técnico, Statistics Department - University of California, Berkeley CA.

Cao, D., Liang, Y., Yan, J., Tan, G., Xu, Q. e Liu, S. (2013). Pydpi: freely available python package for chemoinformatics, bioinformatics, and chemogenomics studies. *Journal of chemical information and modeling*, 51(11):3086–3096. Disponível em https://doi.org/10.1021/ci400127q Acessado em 31/08/2022.

Chen, J., Liu, H., Yang, J. e Chou, K. C. (2012). Classification of endomicroscopic images of the lung based on random subwindows and extra-trees. *IEEE TRANSACTIONS ON BIOMEDICAL ENGINEERING,*, 59(9). Disponível em https://doi.org/10.1109/tbme.2012.2204747 Acessado em 31/08/2022.

Chen, T. e Guestrin, C. (2016). Xgboost: A scalable tree boosting system. Em *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Disponível em https://doi.org/10.1145/2939672.2939785 Acessado em 31/08/2022.

Dubchak, I., Muchnik, I., Holbrook, S. R. e Kim, S. H. (1995). Prediction of protein folding class using global description of amino acid sequence. *Proceedings of the National Academy of Sciences of the United States of America*, 92(19):8700–8704. Disponível em https://doi.org/10.1073/pnas.92.19.8700 Acessado em 31/08/2022.

Désir, C., Petitjean, C., Heutte, L., Salaün, M. e Thiberville, L. (2021). Leukocyte classification based on feature selection using extra trees classifier: atransfer learning approach,. *Turkish Journal of Electrical Engineering and Computer Sciences*, 29(8). Disponível em https://doi.org/10.3906/elk-2104-183 Acessado em 31/08/2022.

Freund, Y. e Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139. Disponível em https://doi.org/10.1006/jcss.1997.1504 Acessado em 31/08/2022.

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189–1232. Disponível em https://doi.org/10.1214/aos/1013203451 Acessado em 31/08/2022.

Galanis, K. A., Nastou, K. C., Papandreou, N. C., Petichakis, G. N., Pigis, D. G. e Iconomidou, V. A. (2021). Linear b-cell epitope prediction for in silico vaccine design: A performance review of methods available via command-line interface. *International journal of molecular sciences*, 22(6). Disponível em https://doi.org/10.3390/ijms22063210 Acessado em 31/08/2022.

Geurts, P., Ernst, D. e Wehenkel, L. (2006). Extremely randomized trees. *Machine Language*, 63(1):3–42. Disponível em https://doi.org/10.1007/s10994-006-6226-1 Acessado em 31/08/2022.

James, G., Witten, D., Hastie, T. e Tibshirani, R. (2013). *An Introduction to Statistical Learning*. Springer. Disponível em https://doi.org/10.1007/978-1-4614-7138-7 Acessado em 31/08/2022.

Junior, C. M. (2020). *Gradient Boosting para a classificação de fácies utilizando perfis de poços*. Trabalho de Graduação (Bacharelado em Geologia) – Universidade de São Paulo.

Kayser, V. e Ramzan, I. (2021). Vaccines and vaccination: history and emerging issues. *Human Vaccines & Immunotherapeutics*, 17(12). Disponível em https://doi.org/10.1080/21645515.2021.1977057 Acessado em 31/08/2022.

Kovenko, V. (2020). Decision trees family. https://machine-learning-and-data-science-with-python.readthedocs.io/en/latest/search.html?q=cite&check_keywords=yes&area=default. Acessado em 31/08/2022.

Kuhn, M. e Johnson, K. (2013). *Applied Predictive Modeling*. Springer. Disponível em https://doi.org/10.1007/978-1-4614-6849-3 Acessado em 31/08/2022.

Liu, T., Shi, K. e Li, W. (2020). Deep learning methods improve linear b-cell epitope prediction. *BioData mining*, 13(1). Disponível em https://doi.org/10.1186/s13040-020-00211-0 Acessado em 31/08/2022.

Lo, Y. T., Shih, T. C., Pai, T. W., Ho, L. P., Wu, J. L. e Chou, H. Y. (2021). Conformational epitope matching and prediction based on protein surface spiral features. *BMC genomics*, 12(Suppl 2):116. Disponível em https://doi.org/10.1186/s12864-020-07303-5 Acessado em 31/08/2022.

Machado, S. L. e Machado, R. D. (1992). *IMUNOLOGIA BÁSICA E APLICADA ÀS ANÁLISES CLÍNICAS*. Auto publicado.

Manavalan, B., Govindaraj, R. G., Shin, T. H., Kim, M. O. e Lee, G. (2018). ibce-el: A new ensemble learning framework for improved linear b-cell epitope prediction. *Frontiers in immunology*, 9:1695. Disponível em https://doi.org/10.3389/fimmu.2018.01695 Acessado em 31/08/2022.

Mortimer, E. A. e Plotkin, S. A. (1998). *Vaccines*. W.B. Saunders.

Mutneja, M., Mohan, C., Long, K., Das, C. e Brockett, R. (2014). *An introduction to Antibodies and their applications*. EMD Millipore.

Nain, Z., Karim, M. M., Sen, M. K. e Adhikari, U. K. (2020). Structural basis and designing of peptide vaccine using pe-pgrs family protein of mycobacterium ulcerans — an integrated vaccinomics approach. *Molecular Immunology*, 120:146–163. Disponível em https://doi.org/10.1016/j.molimm.2020.02.009 Acessado em 31/08/2022.

pandas development team, T. (2020). pandas-dev/pandas: Pandas.

Parvizpour, S., Pourseif, M. M., Razmara, J., Rafi, M. A. e Omidi, Y. (2020). Epitope-based vaccine design: a comprehensive overview of bioinformatics approaches. *Drug Discovery Today*, 25(6):1034–1041.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. e Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Raoufi, E., Hemmati, M., Eftekhari, S., Khaksaran, K., Mahmodi, Z., Farajollahi, M. M. e Mohsenzadegan, M. (2020). Epitope prediction by novel immunoinformatics approach: A state-of-the-art review. *International journal of peptide research and therapeutics*, 26(2):1155–1163. Disponível em https://doi.org/10.1007/s10989-019-09918-z Acessado em 31/08/2022.

Raschka, S. (2014). An overview of general performance metrics of binary classifier systems. *Springer Berlin Heidelberg*. Disponível em https://doi.org/10.13140/2.1.4639.7440 Acessado em 31/08/2022.

Rokach, L. e Maimon, O. (2005). *Decision Trees*, volume 6. Disponível em https://doi.org/10.1007/0-387-25465-X_9 Acessado em 31/08/2022.

Sanchez-Trincado, J. L., Gomez-Perosanz, M. e Reche, P. A. (2008). Ensemble-trees: Leveraging ensemble power inside decision trees. 5255:76–87. Disponível em https://doi.org/10.1007/978-3-540-88411-8_10 Acessado em 31/08/2022.

Sanchez-Trincado, J. L., Gomez-Perosanz, M. e Reche, P. A. (2017). Fundamentals and methods for t- and b-cell epitope prediction. *Journal of Immunology Research*, 2017(2680160). Disponível em https://doi.org/10.1155/2017/2680160 Acessado em 31/08/2022.

Schapire, R. (2013). *Explaining AdaBoost*. Springer Berlin Heidelberg. Disponível em https://doi.org/10.1007/978-3-642-41136-6_5 Acessado em 31/08/2022.

Schatzmann, A. e Pereira, E. Z. (2021). *Classificação de epítopos: Uma abordagem de comparação de características entre datasets*. Trabalho de Graduação (Bacharelado em Informática Biomédica) – Setor de Ciências Exatas, Universidade Federal do Paraná, Curitiba - Paraná.

Sun, P., Ju, H., Liu, Z., Ning, Q., Zhang, J., Zhao, X., Huang, Y., Ma, Z. e Li, Y. (2013). Bioinformatics resources and tools for conformational b-cell epitope prediction. *Computational and mathematical methods in medicine*. Disponível em https://doi.org/10.1155/2013/943636 Acessado em 31/08/2022.

Vasconcellos, P. (2019). Como saber se seu modelo de machine learning está funcionando mesmo. https://paulovasconcellos.com.br/como-saber-se-seu-modelo-de-machine-learning-est%C3%A1-funcionando-mesmo-a5892f6468b. Acessado em 31/08/2022.

Wang, Y., Guo, Y., Pu, X. e Li, M. (2017). A sequence-based computational method for prediction of morfs. *RSC Advances*, 7(31):18937–18945. Disponível em https://doi.org/10.1039/C6RA27161H Acessado em 31/08/2022.

Williams, S. C. P. (2016). Genetic mutations you want. *Proceedings of the National Academy of Sciences of the United States of America*, 113(10):2554–2557. Disponível em https://doi.org/10.1073/pnas.1601663113 Acessado em 31/08/2022.

Yang, Z., Bogdan, P. e Nazarian, S. (2021). An in silico deep learning approach to multi-epitope vaccine design: a sars-cov-2 case study. *Scientific reports*, 11(1):3238. Disponível em https://doi.org/10.1038/s41598-021-81749-9 Acessado em 31/08/2022.

Yao, B., Zhang, L., Liang, S. e Zhang, C. (2012). Svmtrip: a method to predict antigenic epitopes using support vector machine to integrate tri-peptide similarity and propensity. *PloS one*, 7(9). Disponível em https://doi.org/10.1371/journal.pone.0045152 Acessado em 31/08/2022.

Zhang, W., Niu, Y., Zou, H., Luo, L., Liu, Q. e Wu, W. (2015). Accurate prediction of immunogenic t-cell epitopes from epitope sequences using the genetic algorithm-based ensemble learning. *PloS one*, 10(5). Disponível em https://doi.org/10.1371/journal.pone.0128194 Acessado em 31/08/2022.