UNIVERSIDADE DE CAXIAS DO SUL
ÁREA DO CONHECIMENTO DE CIÊNCIAS EXATAS E ENGENHARIAS

ELIZE PERON PIOVESAN

IMPLEMENTAÇÃO E VALIDAÇÃO DE ALGORITMOS DE APLICAÇÃO EM BIOINFORMÁTICA

BENTO GONÇALVES

2024

ELIZE PERON PIOVESAN

IMPLEMENTAÇÃO E VALIDAÇÃO DE ALGORITMOS DE APLICAÇÃO EM BIOINFORMÁTICA

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de bacharel em Engenharia da Computação na Área do Conhecimento de Ciências Exatas e Engenharias da Universidade de Caxias do Sul.

Orientador: Prof. Dra. Scheila de Avila e Silva

BENTO GONÇALVES

2024

ELIZE PERON PIOVESAN

IMPLEMENTAÇÃO E VALIDAÇÃO DE ALGORITMOS DE APLICAÇÃO EM BIOINFORMÁTICA

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de bacharel em Engenharia da Computação na Área do Conhecimento de Ciências Exatas e Engenharias da Universidade de Caxias do Sul.

Aprovado(a) em 26/11/2024

BANCA EXAMINADORA

Prof. Dra. Scheila de Avila e Silva
Universidade de Caxias do Sul - UCS

Prof. Dr. Ricardo Vargas Dorneles
Universidade de Caxias do Sul - UCS

Mestrando Pedro Lenz Casa
Universidade de Caxias do Sul - UCS

RESUMO

O desenvolvimento de algoritmos para resolver problemas na área genética representa um dos grandes desafios da bioinformática, uma ciência interdisciplinar que integra conceitos de exatas e biológicas para analisar dados biológicos complexos. Este trabalho foca na implementação e validação de um algoritmo aplicado à ferramenta BacPP, desenvolvida na Universidade de Caxias do Sul, reconhecida por sua capacidade de classificar promotores bacterianos de maneira independente ao fator regulador associado. Com o objetivo de melhorar a precisão e a confiabilidade na predição de promotores, foi implementado o BacPP-E28, um algoritmo de segunda etapa baseado em aprendizado de máquina, utilizando métricas como precisão, especificidade e sensibilidade para validação. A metodologia incluiu a utilização de dados do RegulonDB, processamento em formato FASTA, e aplicação das regras propostas pelo algoritmo. O BacPP-E28 foi comparado ao BacPP original, demonstrando maior eficiência ao reduzir falsos positivos e negativos, destacando-se na classificação de sequências reguladas pelo fator σ^{28} Os resultados indicam que a integração de algoritmos de validação de segunda etapa baseados em características físico-químicas e padrões de sequências genéticas é uma abordagem promissora, contribuindo significativamente para o avanço da bioinformática.

Palavras-chave: Bioinformática. Classificação de sequências. Validação de algoritmo. Predição de promotores bacterianos.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AC	Acurácia
BAcPP	*Bacterial Promoter Prediction*
DNA	*Deoxyribonucleic Acid*
E. coli	*Escherichia coli*
ESP	Especificidade
FP	Falsos Positivos
FN	Falsos Negativos
Fiocruz	Fundação Oswaldo Cruz
HSV	*Hue - Saturation - Value*
IDE	*Integrated Development Environment*
KNN	*K-Nearest Neighbors*
LRG	*Logistic Regression*
MLP	*Multilayer Perceptron*
NN	*Neural Networks*
PCA	*Principal Component Analysis*
PDCA	*Plan - Do - Check - Act*
PREC	Precisão
RGB	*Red - Green - Blue*
RNA	*Ribonucleic Acid*
ROC	*Receiver Operating Characteristic*
SVM	*Support Vector Machine*
VP	Verdadeiros Positivos
VN	Verdadeiros Negativos
TFN	Taxa Falsos Negativos
TFP	Taxa Falsos Positivos
TSS	*Transcription Start Site*
TVP	Taxa Verdadeiros Positivos

SUMÁRIO

1 INTRODUÇÃO

A bioinformática surgiu a partir da junção da área de ciências exatas e a área de ciências da vida. Ela utiliza conceitos de matemática, física e estatística para analisar dados de origem das ciências da vida e gerar informações com valor biológico. O objetivo principal desta ciência multidisciplinar é organizar, tratar e interpretar dados biológicos, sendo utilizada em diversas áreas de atuação da biotecnologia, como: genética e biologia molecular, saúde, ecologia e agricultura (BIANCO; LIPAY, 2015). Apesar de ser uma ciência emergente, originada em meados de 1980, ela vem crescendo devido ao avanço de tecnologia.

Um dos avanços significativos é a bioinformática translacional, que visa o mapeamento e a análise detalhada do código genético. O principal objetivo dessa área é desenvolver novas terapias e aprimorar o diagnóstico de doenças, promovendo uma abordagem mais personalizada e eficaz na medicina (BARUAH; DEKA; MAHANTA, 2022). Outro avanço destacado neste estudo é a incorporação de conceitos de inteligência artificial e aprendizado de máquina. Segundo Li, Ma e Tian (2022), algoritmos de aprendizado de máquina têm sido utilizados para identificar padrões em dados de sequenciamento, prever interações gene-ambiente e desenvolver modelos preditivos para a toxicidade de compostos químicos. A bioinformática exerce um papel fundamental nas pesquisas relacionadas à área da vida e saúde, auxiliando no entendimento dos mecanismos biológicos associados aos organismos vivos (QUEROZ *et al.*, 2022).

Promotores são regiões específicas do DNA responsáveis por iniciar o processo de transcrição genética, sendo essenciais para a regulação da expressão gênica. A predição de promotores, ou seja, a identificação dessas sequências no genoma, é uma tarefa complexa, pois apesar das sequências apresentarem regiões conservadas, existe um grau de degeneração associado. As características biológicas dos promotores, como a presença de sequências conservadas e a variação na localização dos sítios de ligação de fatores de transcrição, tornam a predição desafiadora, especialmente quando se busca identificar promotores que não seguem os padrões típicos. Técnicas tradicionais de predição, como métodos baseados em análise de motivos, muitas vezes não são suficientes para capturar toda a diversidade das sequências promotoras (SILVA; ECHEVERRIGARAY; GERHARDT, 2011).

O desenvolvimento de algoritmos para auxiliar em aplicações de bioinformática é um dos pilares desta área de conhecimento. Uma prática necessária é a validação desses algoritmos, visto que, eles precisam ser precisos e confiáveis. Uma das formas de validação é a comparação entre o resultado fornecido pelo método e as informações originais (SÁ, 2018). Nesse contexto é possível relacionar Engenharia da Computação e a Bioinformática, uma vez que a computação fornece ferramentas e métodos para auxiliar nas aplicações e a biologia fornece os conceitos necessários para o desenvolvimento desses métodos (QIN *et al.*, 2023).

Diversas ferramentas computacionais foram desenvolvidas ao longo dos anos para melhorar a tarefa de predição de promotores, como o BacPP, que foi desenvolvido na Universidade de Caxias do Sul e é referência para bactérias Gram-negativas, devido a sua capacidade de predição de promotores bacterianos ao considerar múltiplos fatores sigma. Apesar dos avanços, ainda há desafios significativos a serem superados para aumentar a acurácia dessas ferramentas, especialmente no que dizrespeito à redução de falsos positivos e negativos (MARTINEZ, 2023).

Diante desses desafios, este trabalho propõe o desenvolvimento e a validação do BacPP-E28, uma extensão da ferramenta BacPP, focada na classificação de promotores bacterianos regulados pelo fator sigma 28. Utilizando técnicas de aprendizado de máquina e características físico-químicas das sequências de DNA, o BacPP-E28 busca aumentar a precisão na predição, reduzindo falsos positivos e negativos. A solução é projetada para atuar como uma etapa adicional de validação, complementando os métodos existentes na ferramenta original.

O presente estudo inicia apresentando alguns conceitos principais, elencando os tipos de problemas que podem ser tratados computacionalmente e as métricas de validação de classificação que é o tipo de problema que será resolvido com a implementação a ser realizada. Na sequência foram trazidos alguns trabalhos publicados que, apresentam o processo de análise e validação de novos algoritmos, situação que se assemelha à abordagem deste trabalho. Na seção seguinte, a ferramenta BacPP é abordada, com a contextualização da origem da ferramenta e a apresentação de formas para tornar a predição de promotores e não-promotores mais assertiva. A metodologia inclui conceitos de engenharia de software, descrição da base de dados e métricas utilizadas na validação do algoritmo desenvolvido. Por fim, os resultados apresentam a criação e validação do BacPP-E28, evidenciando sua eficácia como solução de segunda etapa na predição de promotores bacterianos. O nome BacPP-E28 reflete suas características principais: 'E' refere-se à estabilidade das sequências analisadas, enquanto '28' indica sua aplicação específica ao fator sigma 28.

2 OBJETIVOS

O objetivo do trabalho é implementar e validar estatisticamente um algoritmo de classificação em segunda etapa na ferramenta BacPP.

2.1 OBJETIVOS ESPECÍFICOS

- Elencar as métricas estatísticas para validação de algoritmo.
- Identificar a metodologia computacional mais aderente para a resolução do problema.
- Definir as estruturas de dados e comandos para melhor execução do algoritmo.
- Analisar o desempenho da tarefa de classificação do algoritmo por meio das métricas elencadas.

3 FUNDAMENTAÇÃO TEÓRICA

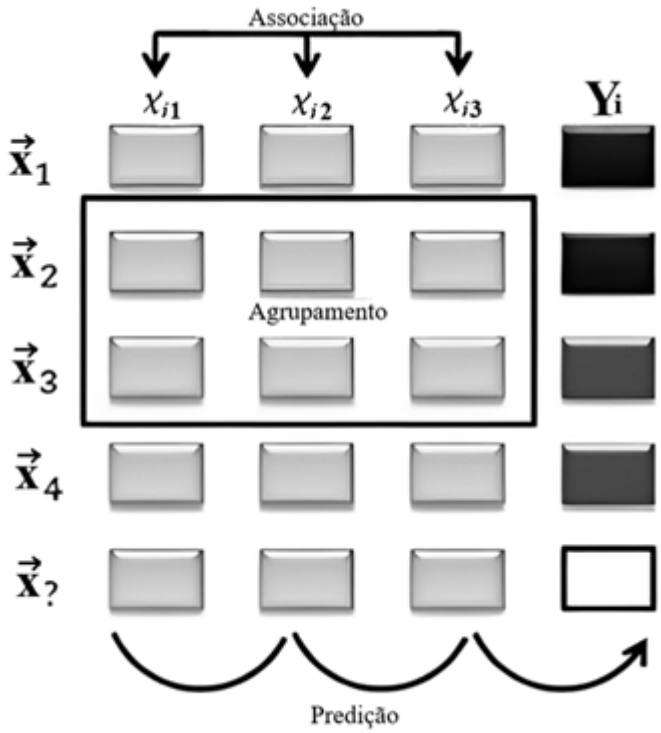
Algoritmo é uma descrição textual de uma solução para um determinado problema, essa descrição é baseada em processos e/ou conceitos básicos já definidos anteriormente (ALVES, 2013). O conceito também pode ser definido como "o caminho de solução para um problema", pois sua principal característica é a presença de uma sequência de regras que serão executadas em uma ordem pré-estabelecida, visando o objetivo final que originou sua criação (SOUZA *et al.*, 2020).

Apesar do conceito de algoritmo ser utilizado principalmente na área da programação, ele também é aplicado em tarefas diárias. Uma receita de bolo é um exemplo que traduz o termo a uma prática comum. Isso porque uma receita é uma sequência de passos que, quando executados corretamente, resultam no alimento desejado. Portanto, para apresentar um novo conjunto de passos ou instruções para a solução de um problema, é necessário identificar o padrão que fundamenta essas instruções, organizá-las de forma lógica e, o mais importante, aplicá-las e validá-las. (FORBELLONE; EBERSPACHER, 2006).

3.1 TIPOS DE PROBLEMAS COMPUTACIONAIS

A definição dos tipos de problemas está diretamente relacionada à mineração de dados, que envolve a obtenção de padrões a partir de um conjunto de dados, com o objetivo de auxiliar nos processos de tomada de decisão frente a problemas em diversas aplicações. As variáveis presentes nos dados determinam as tarefas de mineração de dados, que são divididas em três principais categorias: predição, agrupamento de dados e associação. Essas tarefas são aplicadas conforme a necessidade do problema a ser resolvido, levando em consideração as variáveis envolvidas (SILVA; PERES; BOSCARIOLI, 2016). A Figura 1 exemplifica as três tarefas mencionadas, considerando um mesmo conjunto de dados de forma controlada, garantindo que as condições sejam mantidas consistentemente e evitando interferências externas que possam afetar o resultado final.

Figura 1 – Conjunto de tarefas de mineração de dados



Fonte: Silva, Peres e Boscarioli (2016)

3.1.1 Predição

As tarefas de predição consistem em associar dados específicos retirados de uma base de dados, juntamente com suas principais características, aos rótulos a eles relacionados. O modelo preditivo pode retornar as relações obtidas através de funções ou então organizados em uma estrutura de dados (SILVA; PERES; BOSCARIOLI, 2016).

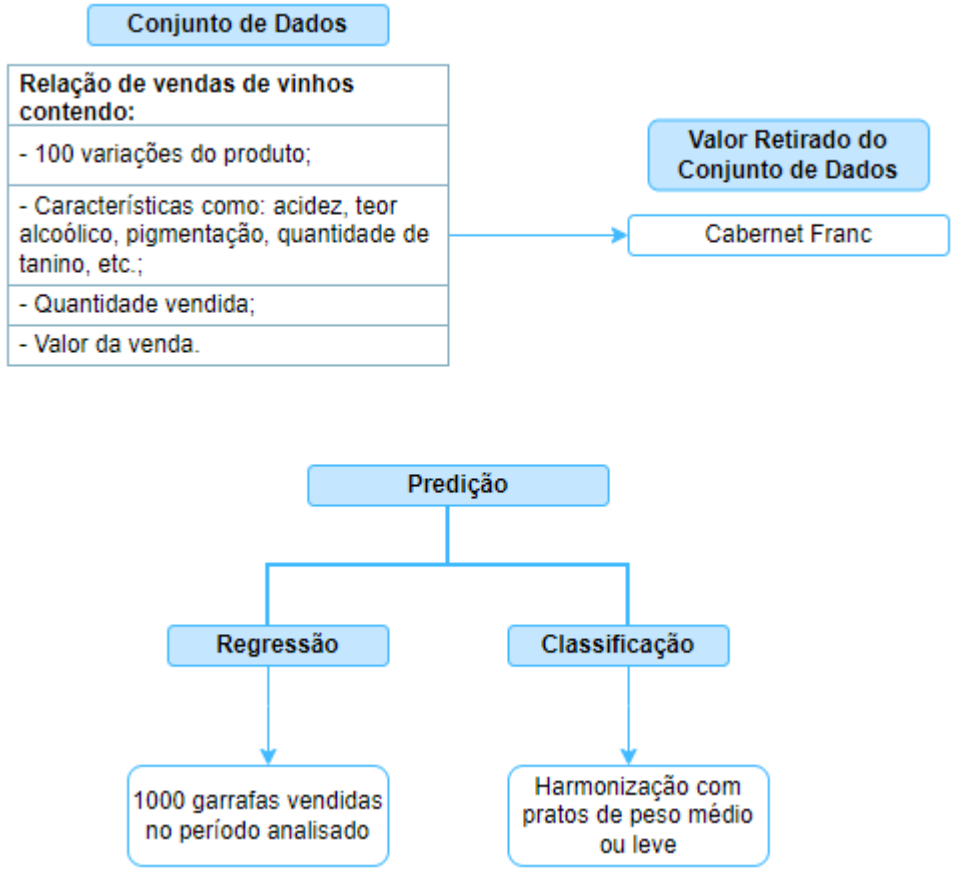
A criação de um modelo preditivo possui duas etapas fundamentais (CASTRO; FERRARI, 2016):

- **Treinamento:** nessa fase, o algoritmo é aplicado a um conjunto de dados no qual as classificações e os valores já estão previamente determinados, ou seja, a saída correspondente a cada entrada é conhecida. O objetivo dessa etapa é desenvolver um modelo capaz de realizar previsões precisas com base nos dados utilizados durante o processo de treinamento.
- **Teste:** nesse momento, realiza-se a validação do funcionamento do algoritmo com dados não utilizados no treinamento, ou seja, quando os valores de saída são desconhecidos. A partir do desempenho obtido é possível definir se o algoritmo pode ser aplicado de forma geral ou não. Essa definição é realizada a partir de métricas específicas para cada uma das tarefas executadas e serão apresentadas na seção 3.2 desse trabalho.

Um valor retirado do conjunto de dados é considerado uma ocorrência em relação à análise que está sendo realizada. Já os rótulos atribuídos a esse valor podem ser exibidos de duas formas: (*i*) identificação da classe pertencente, dentro de um conjunto finito de opções considerando a base analisada; (*ii*) número que o dado analisado está associado, frente ao conjunto contínuo de valores possíveis. Sendo, a primeira forma chamada de Classificação e a segunda de Regressão (FACELI *et al.*, 2021).

Na figura 2 está um exemplo considerando uma base de dados que contempla as vendas de vinhos em determinado período. Nesse conjunto estão descritos alguns atributos como as características da própria bebida, quantidade vendida, etc.

Figura 2 – Exemplificação de Tarefas Preditivas



Fonte: O autor (2024).

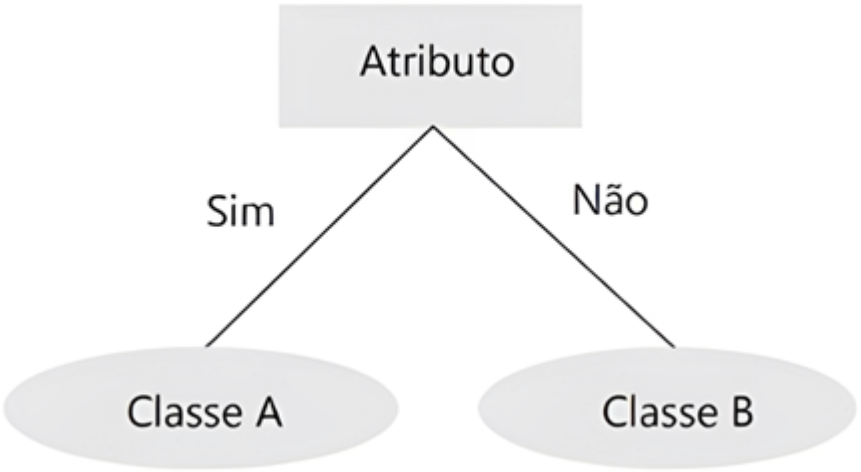
A classificação de um dado consiste em atribuir a ele um ou mais rótulos frente à análise que está sendo realizada, sendo que esse processo ocorre através de um algoritmo treinado a partir de um conjunto de informações previamente rotuladas. Dentro dela, existem duas divisões: classificação binária e classificação multiclasse. O que diferencia elas é a quantidade de opções disponíveis. Enquanto na binária existem apenas duas possibilidades, por exemplo: sim ou não, verdadeiro ou falso, a multiclasse permite uma rotulação mais específica frente ao problema abordado, por exemplo: chuva, sol, calor, frio (SILVA; PERES; BOSCARIOLI, 2016).

De acordo com Faceli *et al.* (2021), a definição formal da tarefa de classificação consiste na função exibida na expressão 3.1 dado um conjunto de observações pares $D = \{(X_i, f(X_i)), i = I, .., n\}$ onde para $f(x_i)$ são atribuídos valores de um conjunto discreto não ordenado.

$$y_1 = f(x_i) \in \{C_1, ..., C_m\} \tag{3.1}$$

Os principais algoritmos de classificação utilizados possuem sua estrutura inspirada em: conhecimento, árvores, distâncias, função e probabilidade (CASTRO; FERRARI, 2016). Na figura 3 é realizada a demonstração gráfica de um modelo de classificação binária com a estrutura baseada em árvores.

Figura 3 – Representação gráfica de um modelo baseado em árvores.



Fonte: Castro e Ferrari (2016)

O problema de regressão, ao contrário do problema de classificação, é utilizado quando a saída desejada é numérica. Os modelos de regressão podem ser divididos em quatro categorias: linear simples, linear multivariado, não linear simples e não linear multivariado. As principais diferenças entre essas categorias estão na equação obtida, pois a linear retorna a equação da reta, enquanto a não linear retorna uma equação exponencial, e na quantidade de atributos considerados para análise, nestes casos a simples considera apenas um atributo, enquanto a multivariada considera dois ou mais (SILVA; PERES; BOSCARIOLI, 2016).

A análise através da regressão linear permite relacionar uma variável dependente com uma ou mais independentes. Em situações nas quais a equação obtida a partir da regressão atinge de forma muito próxima o valor desejado, ainda durante o treinamento do algoritmo, é possível realizar a predição da variável dependente a partir do valor da independente (CASTRO; FERRARI, 2016). Na expressão 3.2 é apresentada a notação formal da regressão linear simples, onde as letras *a* e *b* representam os coeficientes de regressão e definem parâmetros de interceptação no eixo y e inclinação da reta (SILVA; PERES; BOSCARIOLI, 2016).

$$Y = a + bx \tag{3.2}$$

Ainda segundo os autores, ao resolver um problema de regressão é necessário identificar os valores para os coeficientes de regressão, para que as variáveis possam receber os valores do conjunto de dados avaliados e o comportamento do modelo, mantenha-se o mesmo. Nas expressões 3.3 e 3.4, são apresentadas as equações do método dos mínimos quadrados que é o mais indicado ao estimar os coeficientes de regressão.

$$b = \frac{\sum_{i=1}^{n}(x_i - \mu_x)(y_i - \mu_y)}{\sum_{i=1}^{n}(x_i - \mu_x)^2} \tag{3.3}$$

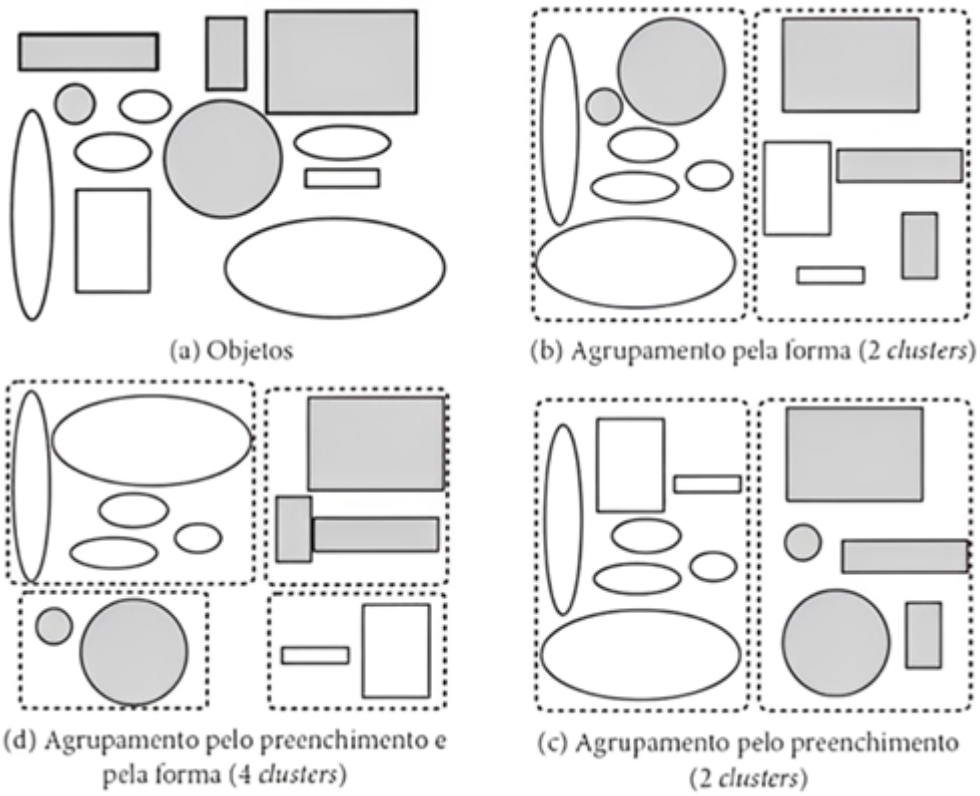
$$a = \mu_y - b\mu_x \tag{3.4}$$

Sendo μ_x a média dos valores de $x_i, x_2, ..., x_n$ e μ_y a média dos valores de $y_i, y_2, ..., y_n$.

3.1.2 Agrupamento

A tarefa de agrupamento, também conhecida como *clustering*, consiste em dividir o conjunto de dados avaliado em pequenos grupos considerando os atributos que possuem em comum. De forma intuitiva é possível compreender que itens pertencentes a um grupo, possuem semelhanças entre si, diferente dos que pertencem a outros grupos (CASTRO; FERRARI, 2016). Diferentemente da tarefa de classificação, essa análise não ocorre baseada nos rótulos associados aos dados analisados, mas sim é fundamentada pelas características descritivas do objeto analisado (SILVA; PERES; BOSCARIOLI, 2016). Na figura 4 é apresentado um conjunto de dados, juntamente de diferentes formas de agrupamentos que podem ser analisadas.

Figura 4 – Exemplo contendo diferentes agrupamentos em um mesmo conjunto de objetos.



Fonte: Faceli *et al.* (2021)

De acordo com Faceli *et al.* (2021), a tarefa de agrupamento não possui uma definição única. As diferentes definições decorrem dos chamados "critérios de agrupamento", os quais auxiliam na escolha da técnica mais adequada para resolver um problema específico. A seguir, são apresentados os principais critérios:

- **Compactação:** baseia-se no princípio de que quanto mais próximos os elementos do grupo estiverem e mais semelhantes forem as características, menor será a variação interna desse agrupamento. Em análises que envolvem agrupamentos esféricos e/ou bem divididos a técnica é estável, porém em análises com um grau maior de complexidade não possui muita eficiência.

- **Encadeamento ou ligação:** baseia-se no princípio de que elementos próximos devem fazer parte do mesmo grupo. Esse formato possui maior assertividade na identificação de agrupamentos de forma irregular ou espalhados pelo conjunto de dados, porém demonstra ser pouco eficaz em cenários onde os dados estão muitos próximos, visto que, o agrupamento pode ocorrer de forma equivocada.

- **Separação Espacial:** considera a distância entre os grupos analisados. Essa técnica permite a validação da qualidade dos agrupamentos. No entanto, seu uso isolado não garante

assertividade nos resultados obtidos. Com isso, esse método costuma ser utilizado em paralelo com outros.

3.1.3 Associação

A associação ou correlação entre características implica na existência de uma relação entre elas. Essa relação pode ser de co-ocorrência, em que os atributos aparecem juntos com frequência, ou de dependência, em que um atributo está relacionado à presença do outro (SILVA; PERES; BOSCARIOLI, 2016).

A tarefa de associação pode ser realizada em diferentes tipos de bases de dados, embora seja aplicada predominantemente em bases de dados transacionais. Um exemplo clássico desse tipo de conjunto de dados é o carrinho de supermercado, pois ao passar cada item comprado na caixa registradora, informações como: produto, quantidade e preço, são armazenadas em uma base de dados, e cada registro desses é um dado transacional (CASTRO; FERRARI, 2016).

Na tabela 1, é apresentado um exemplo de base de dados transacionais. Analisando os dados apresentados, conclui-se que, se uma pessoa compra leite, também compra pão.

Tabela 1 – Exemplo de conjunto de dados transacional.

TID	**Itens**
1	{Leite, pão, açúcar, café, manteiga}
2	{Mamão, banana, maçã}
3	{Leite, pão}
4	{Leite, pão, manteiga, banana}

Fonte: Castro e Ferrari (2016)

Na análise por associação existem dois principais conceitos utilizados para medir os resultados obtidos através dela, sendo eles: Suporte e Confiança (SILVA; PERES; BOSCARIOLI, 2016).

- **Confiança:** a importância e a credibilidade da regra que está sendo avaliada são definidas com base na probabilidade de sua ocorrência, em relação à base analisada. Essa probabilidade é geralmente apresentada em percentual.
- **Suporte:** a frequência com que os itens que estão sendo analisados aparecem na base de dados transacional utilizada é definida. Essa frequência é geralmente apresentada em percentual.

3.2 MÉTRICAS PARA AVALIAÇÃO DE RESULTADOS DE CLASSIFICAÇÃO

A validação do funcionamento de um algoritmo é essencial, independentemente de sua aplicação. Para os algoritmos computacionais, existem métricas específicas que podem ser usadas para essa validação. O tipo de métrica a ser usada depende do tipo de retorno obtido pelo algoritmo (RUSSEL; NORVIG, 2022).

É possível combinar aspectos do problema tratado com técnicas existentes para definir métricas que avaliem o desempenho de algoritmos em novas bases de dados. Além disso, mesmo com um único modelo selecionado, é possível identificar novos padrões de funcionamento a partir de modificações em seus procedimentos livres (FACELI *et al.*, 2021).

Para a definição dos métodos de validação, é necessário levar em consideração uma série de quesitos, como: o problema a ser resolvido, o conjunto de dados que será utilizado na validação, o tipo de resultado que se deseja obter, dentre outros. Cada modelo possui uma maior assertividade e desempenho em situações distintas e é imprescindível o uso de métricas de validação adequadas para obter um desempenho preditivo mais confiável. A matriz de confusão, gerada a partir da etapa de teste de uma abordagem de predição para classificação, é uma ferramenta essencial para a validação do modelo. Ela permite calcular diversas medidas de performance, como acurácia, precisão e sensibilidade, que são fundamentais para a interpretação dos resultados e a avaliação da eficácia do modelo proposto (FACELI *et al.*, 2021).

Na tabela 2, é apresentada uma matriz de confusão para um problema com dois rótulos, ou seja, classificação binária, onde:

- **VP:** corresponde ao número de verdadeiros positivos, ou seja, considerando a classe positiva analisada, quantas amostras foram classificadas corretamente.
- **VN:** corresponde ao número de verdadeiros negativos, ou seja, considerando a classe negativa, quantas amostras foram classificadas corretamente.
- **FP:** refere-se ao número de falsos positivos, ou seja, a quantidade de amostras que pertencem à classe negativa e foram, incorretamente, classificadas como positivas.
- **FN:** refere-se ao número de falsos negativos, ou seja, a quantidade de amostras que pertencem à classe positiva e foram, incorretamente, classificadas como negativas.

Tabela 2 – Matriz de confusão para classificação binária

		Classificação Obtida	
		+	**-**
Classificação	+	VP	FN
Verdadeira	-	FP	VN

A taxa de erro na classe positiva refere-se à taxa de falsos negativos (TFN), que corresponde à classe positiva classificada incorretamente pelo algoritmo, conforme denotado na expressão (3.5).

$$TFN = \frac{FN}{VP + FN} \tag{3.5}$$

A taxa de erro na classe negativa refere-se à taxa de falsos positivos (TFP), que corresponde à classe negativa classificada incorretamente pelo algoritmo, conforme denotado na expressão (3.6).

$$TFP = \frac{FP}{FP + VN} \tag{3.6}$$

A taxa de erro total consiste na razão entre a soma dos valores da diagonal secundária da matriz e todos os elementos dela, conforme denotado na expressão (3.7).

$$ERR = \frac{FP + FN}{VP + VN + FP + FN} \tag{3.7}$$

A acurácia total consiste na razão entre a soma dos valores da diagonal principal da matriz e todos os elementos dela, conforme denotado na expressão (3.8).

$$AC = \frac{VP + VN}{VP + VN + FP + FN} \tag{3.8}$$

A precisão refere-se à proporção dos valores positivos classificados corretamente pelo algoritmo, considerando todos os resultados positivos retornados por ele, conforme denotado na expressão (3.9).

$$PREC = \frac{VP}{VP + FP} \tag{3.9}$$

A sensibilidade refere-se à taxa de verdadeiros positivos (TVP), que corresponde à classe positiva classificada corretamente pelo algoritmo, conforme denotado na expressão (3.10).

$$TVP = \frac{VP}{VP + FN} \tag{3.10}$$

A especificidade também conhecida como *Recall* ou revocação, refere-se à taxa de verdadeiros negativos, que corresponde à classe negativa classificada corretamente pelo algoritmo, conforme denotado na expressão (3.11).

$$ESP = \frac{VN}{VN + FP} \tag{3.11}$$

A F1- *Score* refere-se à média harmônica simples entre precisão e especificidade, conforme denotado na expressão (3.12).

$$F1 - \textit{Score} = 2 * \frac{\text{Precisão} * Especificidade}{\text{Precisão} + Especificidade} \tag{3.12}$$

3.3 TRABALHOS RELACIONADOS

Nesta seção, são apresentados trabalhos que abordam a validação de algoritmos em diferentes aplicações e tipos de problemas. Os artigos tratam de aplicações na agricultura, biologia e medicina.

É apresentado por Silva *et al.* (2023) a validação de uma ferramenta criada para auxiliar na classificação de imagens. O algoritmo desenvolvido com base em visão computacional e computação embarcada, tem como objetivo otimizar os processos produtivos da agricultura e reduzir perdas por meio do controle de pragas e doenças.

O estudo de caso utilizado na criação do algoritmo consiste na análise de imagens de casca de laranja, visto que diversas doenças que acometem a fruta possuem características evidentes já na sua superfície. Conforme mencionado pelos autores, com base em informações da Associação Nacional dos Exportadores de Sucos Cítricos, a laranja é uma das frutas mais produzidas a nível mundial e está vinculada a diversos processos industriais, como por exemplo no ramo de produtos de limpezas. Com isso, o controle de qualidade do fruto precisa ser altamente eficiente, frente às doenças que podem afetar de forma significativa sua produção. As principais doenças que afetam a laranja são: Cancro Crítico, Pinta Preta e *Greening* Crítico, conforme representado na figura 5.

Figura 5 – Principais doenças que afetam a laranja.

Fonte: Adaptado de Silva *et al.* (2023)

Atualmente, a inspeção de frutos é frequentemente realizada de forma manual, um processo que pode ser oneroso e, em alguns casos, ineficaz, dependendo do momento da verificação. A identificação tardia de doenças pode impactar negativamente a qualidade da laranja e, consequentemente, todo o sistema produtivo.

Do ponto de vista computacional, as imagens consistem em uma disposição de *pixels*, e sua representação é determinada por um sistema de cores que é selecionado com base na eficácia apropriada para a aplicação em questão. No contexto deste estudo de caso específico, optou-se pelo método HSV (*Hue, Saturation, Value*), no qual cada cor é descrita em termos de seus valores de matiz, saturação e brilho distintos. Esse sistema foi adotado devido à sua menor sensibilidade a flutuações na iluminação, o que, por sua vez, permite a extração de informações essenciais, como textura, formato e intensidade de cores. Tais informações desempenham um papel fundamental no fornecimento de dados para os algoritmos empregados na construção de modelos de Inteligência Artificial.

O método proposto, utilizou técnicas de visão computacional para coletar aspectos das imagens avaliadas. Os dados coletados da imagem foram armazenados em um vetor de recursos, para posteriormente serem utilizados no treinamento do algoritmo. Notavelmente, tais informações foram extraídas do canal H (Matiz) das imagens. Para realizar essa extração, uma etapa preliminar se faz necessária: a conversão da imagem do seu formato padrão RGB para o espaço de cores HSV (Matiz, Saturação, Valor). Conforme mencionado pelos autores, o algoritmo pode ser dividido em três etapas principais: (*i*) carregamento da imagem; (*ii*) extração do vetor de características da imagem; (*iii*) classificação da imagem.

Na tabela 3, estão descritos os dados utilizados no treinamento do algoritmo.

Tabela 3 – Base de dados utilizada para treinar o algoritmo

Classe	**Quantidade de Imagens**
Cancro	1002
Fresca	1446
Pinta Preta	1030
Greening	1845

Fonte: Adaptado de Silva *et al.* (2023)

A validação dos algoritmos foi realizada por meio das métricas de classificação precisão, *recall* e F1-*score*. Na tabela 4 é possível visualizar os resultados obtidos para cada algoritmo utilizado.

Além disso, a acurácia foi avaliada para cada método. O modelo *Support Vector Machine* (SVM) atingiu uma acurácia de 96,08%, o *k-Nearest Neighbors* (KNN) alcançou 96,00%, o Random Forest obteve 96,26% e o *Multilayer Perceptron* (MLP) apresentou a maior acurácia, com 98,13%. De forma geral, o algoritmo MLP demonstrou o melhor desempenho, alcançando a maior precisão (acurácia) entre os métodos avaliados.

Tabela 4 – Comparação entre métodos diferentes

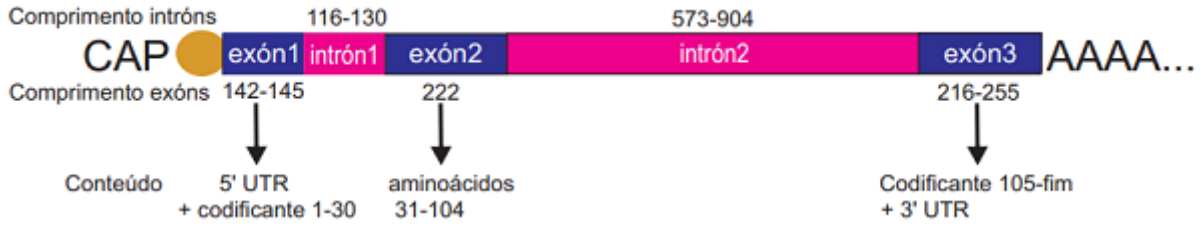
Métodos	Variável	Precisão	*recall*	F1-*score*
SVM	Frescas	1.00	1.00	1.00
	Cancro	0.91	0.88	0.89
	Greening	1.00	1.00	1.00
	Pinta Preta	0.89	0.92	0.91
KNN	Frescas	1.00	0.98	0.99
	Cancro	0.88	0.92	0.90
	Greening	1.00	1.00	1.00
	Pinta Preta	0.90	0.88	0.89
Random Forest	Frescas	1.00	1.00	1.00
	Cancro	0.94	0.87	0.90
	Greening	1.00	1.00	1.00
	Pinta Preta	0.88	0.95	0.91
MLP	Frescas	1.00	1.00	1.00
	Cancro	0.94	0.96	0.95
	Greening	1.00	1.00	1.00
	Pinta Preta	0.96	0.94	0.95

Fonte: Adaptado de Silva *et al.* (2023)

Embora a aplicação final não seja exatamente a mesma, a estrutura do algoritmo está alinhada com o que será desenvolvido neste trabalho. Isso ocorre porque a proposta é realizar uma classificação a partir de dados fornecidos de padrões previamente conhecidos e realizar a validação do mesmo, também, por meio de métricas classificatórias. A partir desse trabalho, foi possível observar a importância de se utilizar mais de uma métrica classificatória na validação de algoritmos, para que se verifique se o método avaliado atenderá ao resultado esperado em diferentes aspectos.

No artigo "*BASiNET—BiologicAl Sequences NETwork: a case study on coding and non-coding RNAs identification*"de Ito *et al.* (2018) é apresentado um modelo capaz de extrair características de RNAs codificantes e não-codificantes, desenvolvido através de redes complexas podendo ser ampliado para outras aplicações biológicas associadas a DNA e sequências de aminoácidos. Na figura 6, está sendo apresentada a estrutura de um RNA.

Figura 6 – Estrutura de um RNA.



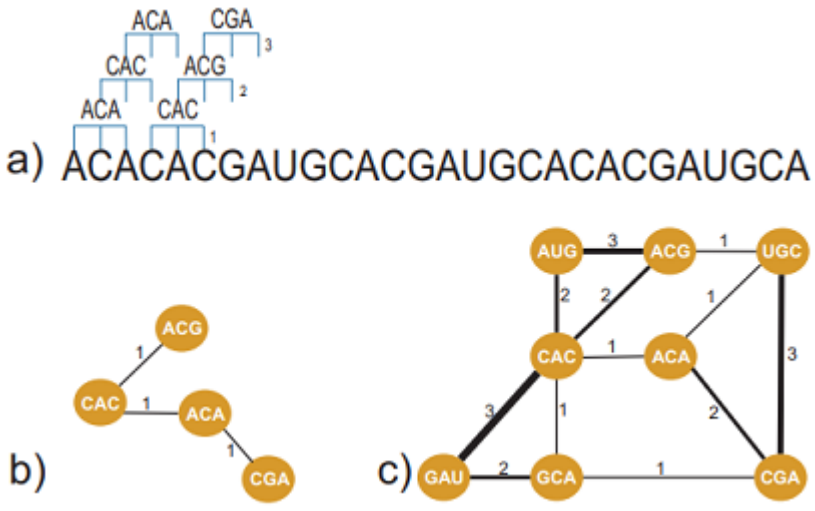
Fonte: Ito *et al.* (2018)

Métodos computacionais são frequentemente utilizados para auxiliar na classificação de RNAs codificantes e não-codificantes. No entanto, a tarefa apresenta um nível de complexidade para os profissionais da área, pois depende de um significativo volume de dados anotados, da existência de ruídos originados do sequenciamento e da necessidade de indicadores seguros de acurácia. O método proposto pelos autores foi desenvolvido utilizando métricas de redes complexas para a extração de características de sequências biológicas de RNAs. Esse tipo de rede foi escolhido por apresentar maior precisão e flexibilidade, o que o torna bastante indicado para a classificação de sequências codificantes e não-codificantes.

A ferramenta projetada representa a sequência de RNA como um grafo, no qual os vértices representam os segmentos de nucleotídeos da sequência avaliada e as arestas representam as relações estruturais entre os segmentos. A frequência de ocorrência de segmentos próximos no RNA é utilizada para avaliar a relevância das arestas. A transformação de uma sequência em grafo é feita a partir de dois parâmetros de configuração: o tamanho da palavra (*Word Size* - WS), que representa a quantidade de nucleotídeos, e o tamanho do passo (*Step Size* - ST), que define as conexões entre eles. Após essa primeira análise com a representação em grafos, é realizado um refinamento através da aplicação de limites para diminuir arestas menos espessas.

Tanto na fase de montagem dos grafos iniciais, como no refinamento com a geração de subgrafos, são coletadas algumas métricas de redes complexas, sendo elas: proximidade, grau, grau máximo, grau mínimo, intermediação, coeficiente de *clustering*, caminho mínimo médio, desvio padrão, *motif* de tamanho 3 e *motif* de tamanho 4. Na figura 7, temos uma sequência RNA representada em grafo.

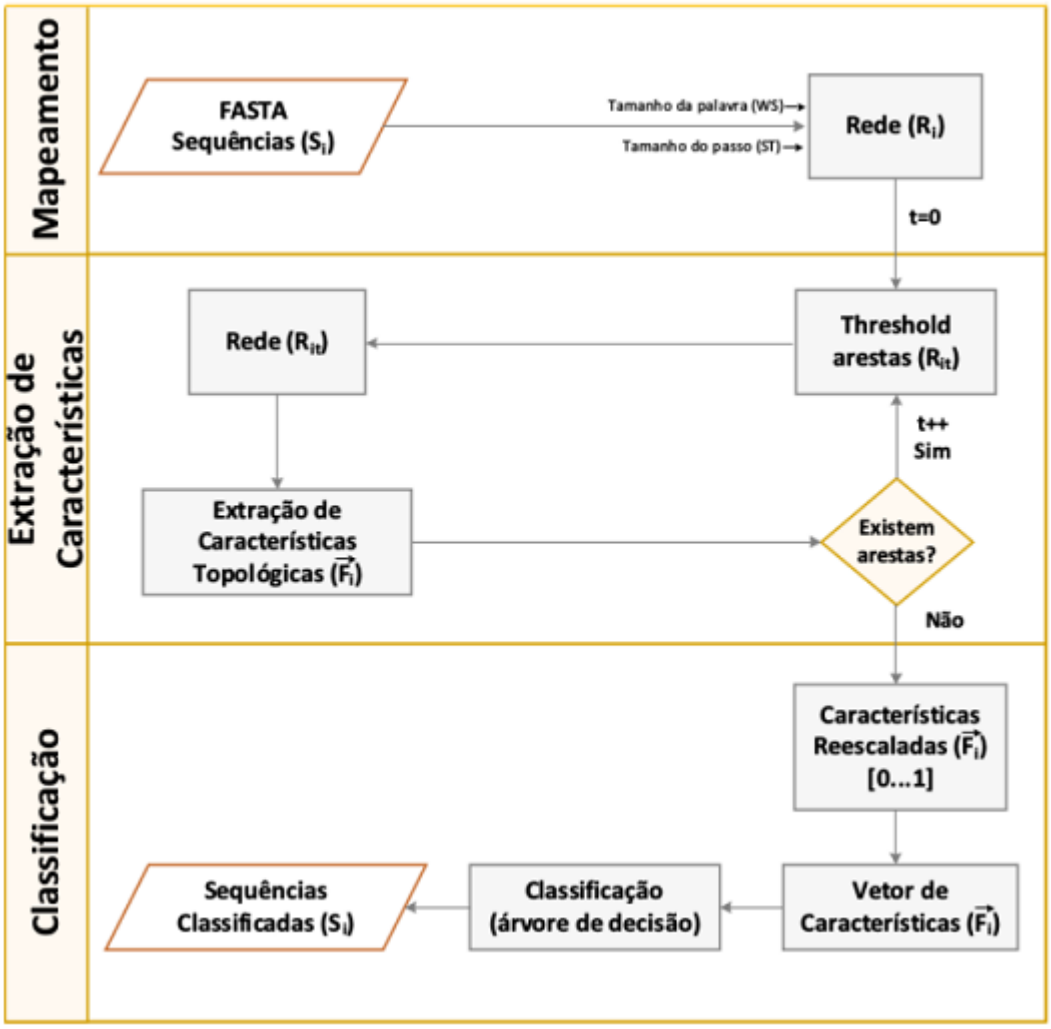
Figura 7 – Grafo de uma sequência com tamanho de palavra 3 e tamanho de passo 1. Considerando: (a) sequência RNA; (b) grafo das três primeiras iterações do mapeamento; (c) grafo completo da sequência.



Fonte: Ito *et al.* (2018)

Segundo os autores, o método pode ser dividido em três grandes etapas principais: mapeamento, extração de características e classificação. Na Figura 8, cada etapa é apresentada juntamente com os passos executados.

Figura 8 – Fluxograma de funcionamento do método desenvolvido.



Fonte: Ito *et al.* (2018)

A linguagem R foi escolhida para o desenvolvimento da ferramenta. Além das métricas específicas de redes complexas, foi utilizada a métrica de classificação acurácia para validar o desempenho e funcionamento do método proposto e compará-lo a outros semelhantes. Na Tabela 5 é possível observar a acurácia obtida para algumas espécies e duas classes de RNA distintas utilizando o algoritmo proposto pelos autores.

Tabela 5 – Acurácia obtida a partir do algoritmo *BASiNET*

Espécie	Classe de RNA	Acurácia
Mus musculus	mRNA	100.00
	ncRNA	99.9
Danio rerio	mRNA	100.00
	ncRNA	98.9
Xenopus tropicalis	mRNA	100.00
	ncRNA	100.00
Bos taurus	mRNA	100.00
	ncRNA	98.9
Pan troglodytes	mRNA	100.00
	ncRNA	99.8
Sus scrofa	mRNA	99.9
	ncRNA	99.6
Macaca mulatta	mRNA	100.00
	ncRNA	100.00

Fonte: Adaptado de Ito *et al.* (2018)

O artigo em questão possui uma aplicação final muito semelhante ao presente trabalho, pois o algoritmo receberá como entrada um arquivo do tipo *Fasta* com uma sequência de ácido nucleico desejado e deverá retornar a classificação da sequência. O método desenvolvido pelos autores foi fundamentado em redes complexas, enquanto o algoritmo empregado no presente trabalho é baseado em redes neurais. A diferença entre as redes utilizadas impactará na forma como o algoritmo será treinado, como os dados serão representados e as métricas que serão utilizadas na validação, mas o processo executado é muito semelhante sendo: *(i)* desenvolvimento do algoritmo; *(ii)* utilização de dados já conhecidos para o treinamento do método; *(iii)* confirmação dos resultados apresentados pelo algoritmo através de pelo menos três métricas de validação.

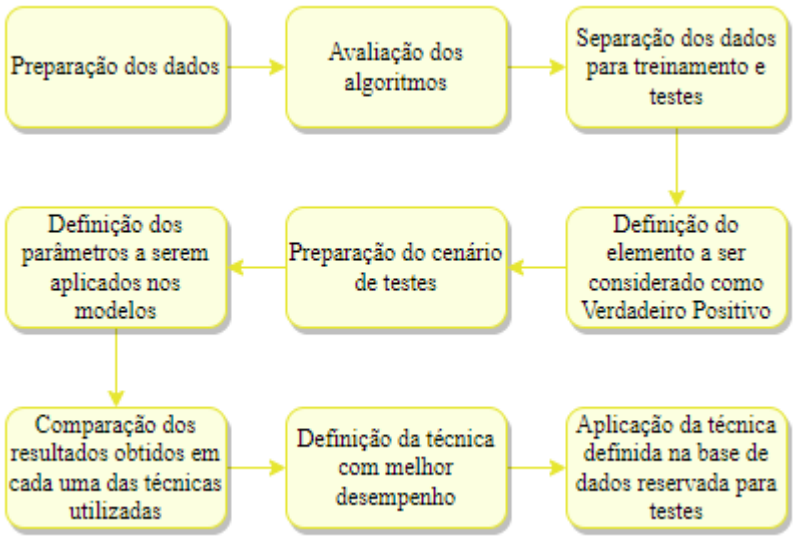
No artigo publicado por Rosa *et al.* (2023), os autores apresentaram um estudo envolvendo a utilização das técnicas de aprendizagem de máquina para auxiliar na identificação de fatores importantes em episódios de doenças ocupacionais. O objetivo principal é, por meio da predição, identificar e classificar os principais fatores que causam dermatite ocupacional, a fim de apontar ações mais eficientes visando a segurança dos profissionais.

A área que trata de Segurança e Saúde no Trabalho age para prevenir acidentes de trabalho e doenças ocupacionais através do aperfeiçoamento de técnicas, mecanismos e estratégias para reduzir essas situações. Dessa forma, é importante que esse desenvolvimento ande junto com os avanços tecnológicos buscando aplicar métodos que possuam eficácia comprovada. Com o volume de dados gerados pelas organizações, o uso de técnicas e ferramentas originadas do aprendizado de máquina e inteligência artificial vem evoluindo cada vez mais. Diante disso, o estudo desenvolvido pelos autores consiste em avaliar uma série de fatores para a montagem de

um modelo computacional que auxilie na classificação rápida e eficaz de doenças ocupacionais. Para isso, foi realizada uma comparação de técnicas de aprendizagem de máquina alimentadas com informações extraídas de um banco de dados da Fundação Oswaldo Cruz (Fiocruz) que contém registros de trabalhadores que procuraram atendimento na instituição com algum sintoma que indicava doenças ocupacionais relacionadas à pele. Para a construção desse modelo foram utilizados alguns aspectos importantes, como: sexo, profissão, etnia, idade, etc.

Para a validação, os autores utilizaram dados de 616 trabalhadores originados no período de 2000 a 2014. Ao longo da etapa de preparação dos dados foram eliminados do conjunto 56 registros, devido a não ser possível definir se os sintomas apresentados estavam relacionados ao trabalho. Os autores utilizaram o *software* R versão 4.6.1 e avaliaram doze técnicas diferentes na gama de problemas de regressão e classificação, sendo estas definidas de acordo com a disponibilidade do programa utilizado. Do conjunto mencionado, cerca de 80% foi aplicado no treinamento dos algoritmos e os outros 20% utilizado nos testes dos mesmos. Para cada técnica utilizada aplicou-se dez métricas para validação de desempenho sendo elas: erro, acurácia, precisão, sensibilidade, especificidade, F1-*Score*, detecção e prevalência. A metodologia utilizada pelos autores está dividida em nove etapas principais, conforme apresentado na Figura 9.

Figura 9 – Metodologia realizada pelos autores



Fonte: Adaptado de Rosa *et al.* (2023)

A abordagem realizada por Rosa *et al.* (2023), realiza a comparação de diversas técnicas e avalia seu desempenho por meio de diferentes métricas. No entanto, optou-se por incluir apenas as técnicas e métricas que são relevantes para o presente trabalho. Na Tabela 6, são apresentados os valores de desempenhos obtidos, considerando técnicas e métricas escolhidas. Na Figura 10, está sendo apresentada a curva ROC, uma representação gráfica que relaciona a taxa de verdadeiros positivos com a taxa de falsos positivos, gerada com objetivo de visualizar

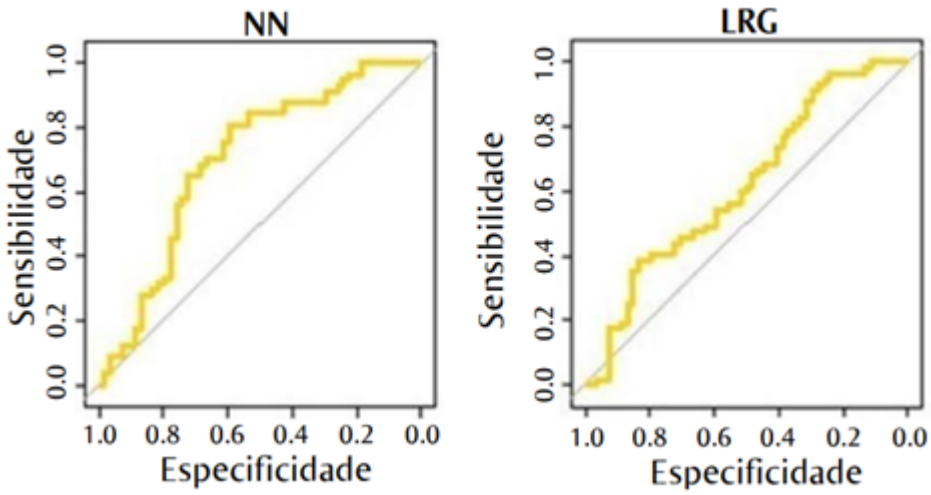
o comportamento de cada método utilizado frente aos acertos nas predições realizadas, obtida para as duas técnicas escolhidas.

Tabela 6 – Técnicas avaliadas e os valores de desempenho obtidos.

Métrica	**Redes Neurais**	**Regressão Logística**
Acurácia	0,694	0,55
Especificidade	0,574	0,5
Precisão	0,667	0,557
F1-*Score*	0,73	0,576
Erro	0,306	0,451

Fonte: Adaptado de Rosa *et al.* (2023)

Figura 10 – Curva ROC obtidas para as técnicas Redes Neurais (NN) e Regressão Logística (LRG).



Fonte: Adaptado de Rosa *et al.* (2023)

Apesar do artigo em questão não possuir aplicação em genética, ele foi relacionado devido ao fato de avaliar os resultados de diferentes métodos computacionais e identificar a Rede Neural como a técnica com melhor desempenho, conceito esse aplicado no desenvolvimento da ferramenta BacPP, mencionado na Seção 3 desse trabalho e que será a aplicação final deste traballho. Além disso, o desempenho dos algoritmos avaliados foi medido utilizando uma gama variada de métricas e algumas delas serão utilizadas na validação do método desenvolvido.

4 BACPP

Esta seção descreve o algoritmo que será implementado e validado neste trabalho.

4.1 SITUAÇÃO ATUAL

O BacPP (*Bacterial Promoter Prediction*) é uma ferramenta computacional desenvolvida na Universidade de Caxias do Sul, conforme descrito por Silva (2011), e que se mantém em evolução até hoje. Seu objetivo é realizar a predição de promotores bacterianos a partir de uma sequência de nucleotídeos inserida pelo usuário, podendo ser realizada manualmente ou através da importação de um arquivo. Independente de como a sequência for inserida no programa ela deve estar no formato *Fasta*, ou seja, arquivo que possui linhas de até 80 caracteres contendo a sequência a ser analisada, sendo cada entrada iniciada por um cabeçalho indicado pelo caractere ">" em conjunto com uma descrição e a sequência respectiva em uma nova linha (MARTINEZ, 2023).

Os promotores exercem função central na expressão gênica, regulam o início da transcrição do gene ficando na mesma molécula de DNA que ele. As células bacterianas são reguladas por fatores sigma (σ), que ajustam a expressão gênica de acordo com as mudanças ambientais. A bactéria E. *coli*, que é um organismo modelo, tem os fatores reguladores divididos em seis grupos (σ^{24}, σ^{28}, σ^{32}, σ^{38}, σ^{54}, σ^{70}), cada um responsável por uma função específica.

Os fatores σ^{24} e σ^{32} estão associados à resposta ao choque térmico. O fator σ^{28} está relacionado à expressão de genes flagelares durante o crescimento normal. O fator σ^{54} está relacionado ao metabolismo do nitrogênio, e o σ^{70} é relacionado à maior parte da atividade transcricional de manutenção celular. (SILVA; GERHARDT; ECHEVERRIGARAY, 2011). Por outro lado, o fator σ^{38} desempenha um papel crucial como principal regulador da resposta geral ao estresse na *Escherichia coli* (TRIPATHI; ZHANG; LIN, 2014). Os promotores são sequências que possuem duas regiões com maior conservação dos seus nucleotídeos. Com exceção do fator regulador σ^{54}, as regiões se localizam a -10 e -35 nucleotídeos em relação ao local de início da transcrição (TSS). Essas regiões são mais preservadas pois são os pontos que o fator σ detecta para iniciar a transcrição. O TSS é definido como a posição +1 de um promotor, antes dele a sequência é um promotor e depois é parte de um gene.

O reconhecimento de promotores para cada fator costuma ocorrer com um padrão nucleotídico diferente nessas regiões, por isso, o desempenho da análise global fica prejudicado, ou seja, não é possível utilizar um único padrão para estimar todos promotores. A predição com maiores índices de exatidão é do fator σ^{70}, dessa forma, o promotor σ^{70} é utilizado como sequência base e isso causa distorção na análise global, resultando em uma taxa intolerável de falsos negativos (SILVA; GERHARDT; ECHEVERRIGARAY, 2011).

Considerando esse cenário, o estudo para desenvolvimento da ferramenta BacPP foi iniciado, visando melhorar o desempenho da bioinformática na classificação de sequências de DNA em promotor ou não-promotor. Para isso, foi utilizada a técnica de Redes Neurais devido à sua eficácia no reconhecimento de padrões degenerados, imprecisos e incompletos disponíveis nas sequências avaliadas. Com ela, também é possível retirar diretrizes de redes já treinadas que são úteis na criação de novas técnicas relevantes no ponto de vista biológico, considerando os parâmetros de entrada. A ferramenta BacPP é fundamentada em regras extraídas do processo de aprendizagem de Redes Neurais para sequências promotoras dependentes de todos os fatores reguladores apresentados anteriormente. A partir das informações obtidas dessa extração foi realizada uma ponderação entre elas visando melhorar o desempenho para classificar os promotores conforme seu fator σ.

Para o treinamento da ferramenta foi utilizado um conjunto de 1034 sequências de promotores extraídas do banco de dados *RegulonDB* (Gama-Castro *et al.* (2008))[1], variando o fator σ regulador, conforme mostra a Tabela 7, e a mesma quantidade de sequências aleatórias utilizadas como não-promotores para garantir a especificidade da ferramenta.

Tabela 7 – Sequências utilizadas no treinamento agrupadas pelo fator σ regulador

Fator σ	**Nº Sequências**
σ^{24}	69
σ^{28}	21
σ^{32}	71
σ^{38}	99
σ^{54}	38
σ^{70}	740

Fonte: Silva, Echeverrigaray e Gerhardt (2011)

Para cada sequência promotora foram realizadas simulações de redes neurais e os nucleotídeos foram codificados em quatro dígitos binários, como exemplificado na Tabela 8. Identificou-se que uma sequência é promotora se o resultado obtido estiver entre 0,5 e 1. Para validar os resultados obtidos pelas redes neurais, os autores utilizaram três das métricas classificadoras mencionadas na Seção 3.2 do presente trabalho, sendo elas: precisão, especificidade e sensibilidade.

[1] https://regulondb.ccg.unam.mx/

Tabela 8 – Codificação binária dos nucleotídeos

Nucleotídeo	Codificação
A	0100
T	1000
C	0001
G	0010

Fonte: Adaptado de Silva, Echeverrigaray e Gerhardt (2011)

A extração das regras das redes neurais ocorreu a partir da técnica Fuzzy Automatically Generated Neural Inferred System, que consiste em separar uma função sigmoide em três partes. Ela se baseia em trocar a função de ativação por um conjunto de segmentos lineares, utilizando a relação apresentada na equação (4.1).

$$f_A(a_j) \sim i[F_i(a_j)(p_i a_j + q_i)] \tag{4.1}$$

Onde, $f_A(a_j)$ é a função original não linear; a_j é o sinal de ativação, obtido através da soma ponderada dos vetores da entrada; F_i é a função responsável por atribuir cada valor de a_j aos respectivos segmentos lineares; $(p_i a_j + q_i)$ são segmentos lineares. Os resultados obtidos podem ser apresentados através de uma equação linear simples, como a demonstrada na expressão (3.2) da Seção 3.1.1, onde: x é a entrada dos dados; y representa a saída das redes neurais; os coeficientes da equação equivalem aos nucleotídeos da sequência.

O propósito da ferramenta BacPP é ponderar a pontuação obtida a partir da extração das regras de redes neurais (NN) para cada sequência dependente de um fator σ regulador, sendo estas utilizadas para definir e identificar promotores de acordo com o respectivo regulador. Os pesos utilizados na ponderação variam de - 10 a + 10. A partir da ponderação realizada, as sequências promotoras passaram novamente pela regra para então definir um valor para cada nucleotídeo e, com isso, definir um ponto de corte entre promotores e não-promotores, sendo que devido à variação dos fatores reguladores, cada um possui um ponto de corte distinto. Na Tabela 9, são apresentados os valores obtidos para cada pontuação avaliada.

Na Figura 11, está o perfil obtido para promotores regulados pelo fator σ^{32}, onde é possível observar dois fragmentos parcialmente preservados sendo um deles na posição de 7 a 15 e o outro entre as posições 28 e 3. Na Figura 12, temos o histograma obtido para todas sequências dependentes do fator σ^{32}, sendo consideradas promotoras e não-promotoras. Com ele é possível identificar o ponto de corte para classificação correta das sequências pertencentes a esse grupo, sendo este identificado no valor 20 de ponderação.

Tabela 9 – Melhor conjunto de pesos para os resultados da classificação BacPP

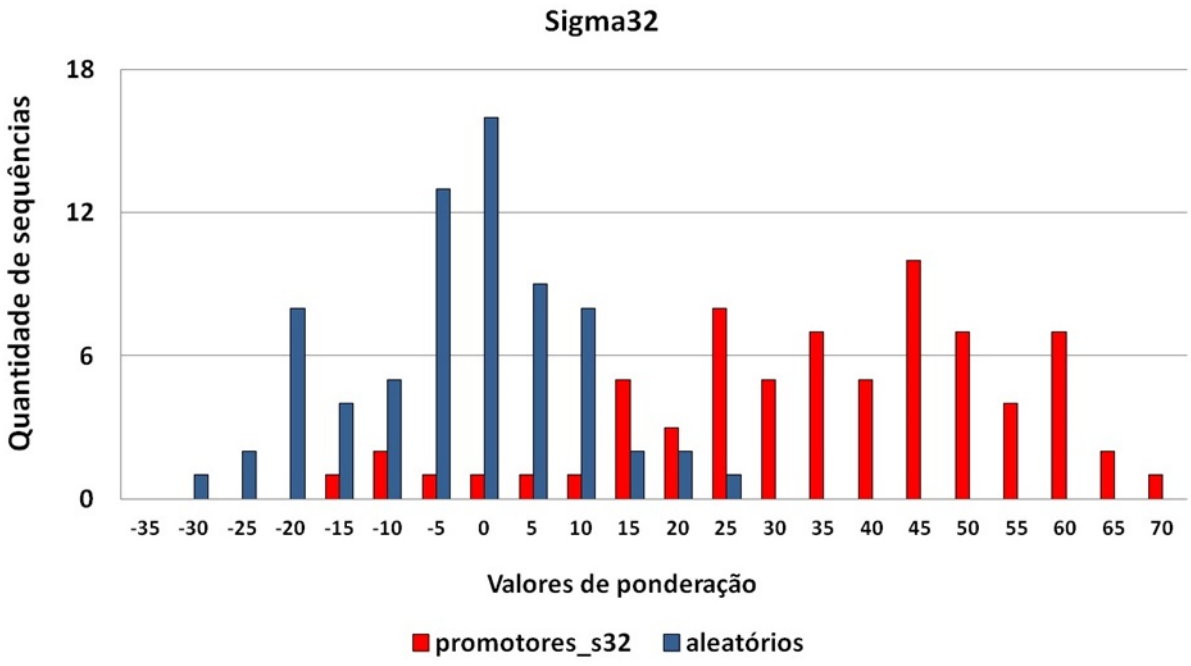
Pontuação NN	Valor Obtido
Maior que 0,6	+6
Entre 0,5 e 0,59	+4
Entre 0,4 e 0,49	+2
Entre 0,3 e 0,39	+1
Entre 0,2 e 0,29	0
Entre 0,1 e 0,19	-1
Menor que 0,1	-3

Fonte: Silva, Echeverrigaray e Gerhardt (2011)

Figura 11 – Expressão obtida para promotores regulados pelo σ^{32}

Fonte: Adaptado de Silva, Echeverrigaray e Gerhardt (2011)

Figura 12 – Histograma com valores obtidos para σ^{32}



Fonte: Adaptado de Silva, Echeverrigaray e Gerhardt (2011)

A ferramenta apresentou resultados comparáveis com a literatura, visto que obteve maior precisão, especificidade e sensibilidade do que as redes neurais que originaram as regras (SILVA; GERHARDT; ECHEVERRIGARAY, 2011). Na Tabela 10, são apresentados os resultados

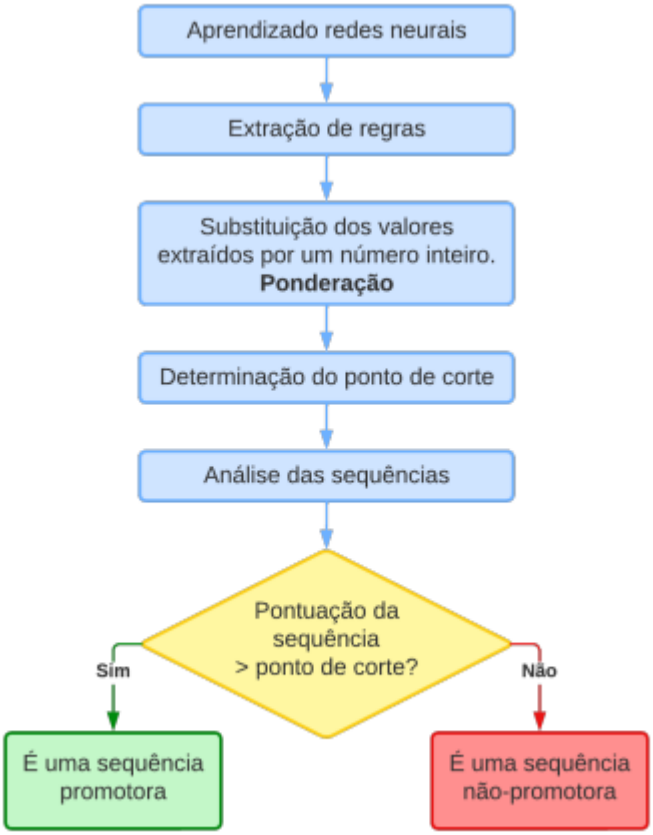
de aprendizagem da ferramenta, para cada fator, considerando a bactéria *E. coli*. O fluxo de funcionamento da ferramenta é apresentado, de forma objetiva, na Figura 13.

Tabela 10 – Resultados obtidos para a bactéria *E. coli*

Fator σ	**% Acurácia**	**% Sensibilidade**	**% Especificidade**
σ^{24}	86,9	95,6	78,2
σ^{28}	92,8	90,4	95,2
σ^{32}	91,5	92,9	90,1
σ^{38}	89,3	83	93,9
σ^{54}	97	100	94,11
σ^{70}	83,6	85,4	81,8

Fonte: Martinez (2023)

Figura 13 – Funcionamento do BacPP.



Fonte: Conci (2023)

A ferramenta era inicialmente um *script* em Python, executado localmente por meio de linhas de comando. Em 2014, visando uma melhor experiência do usuário, a ferramenta passou a contar com uma interface gráfica por meio de um *website* [2] (CONCI, 2023). Ao acessar o *link* mencionado, o usuário será direcionado à página exibida, onde deverá realizar o *login* para

[2] http://www.bacpp.bioinfoucs.com/home

poder inserir a sequência desejada. Após o *login*, a seção BacPP é liberada para a introdução da sequência de DNA. A inserção pode ser realizada por meio da caixa de texto disponível ou pela importação de um arquivo do tipo *Fasta*, vale ressaltar que a ferramenta exige que os dados de entrada sigam um padrão. Após, deve ser selecionado o fator sigma desejado, bem como a forma que o resultado deve ser gerado (em tela ou arquivo) e, por fim o usuário deverá clicar em *run* para executar a ferramenta.

O BacPP representou um grande avanço na classificação de promotores, em comparação aos métodos existentes até então. No entanto, é necessário continuar a evoluir o método para melhorar o seu desempenho. Na seção a seguir, são abordadas as evoluções propostas.

4.2 IMPLEMENTAÇÕES FUTURAS

A predição de promotores, área da biotecnologia que trabalha na concepção de ferramentas que auxiliam na identificação de sequências de DNA promotoras, segue aberta para estudos. Diversas tentativas já foram realizadas na abordagem do assunto, porém a classificação de sequências em promotoras ou não-promotoras ainda é uma área que merece investigação, pois alguns tipos de sequências apresentam características que as tornam menos estáveis (DALL'ALBA, 2018).

Os métodos computacionais desenvolvidos a partir do aprendizado de máquina com o objetivo de classificar sequências promotoras ou não-promotoras são afetados devido a alguns traços do conjunto de dados avaliado, como por exemplo: (*i*) o fato da sequência de nucleotídeos não ser idêntica em 100% por conta de deterioração ou combinações distintas de aspectos; (*ii*) o fato de um promotor possuir um tamanho considerado pequeno aumentando a possibilidade de confusão com outras regiões do DNA. Diante disso, é comum as ferramentas classificarem falsos positivos e falsos negativos (CASA *et al.*, 2022).

No artigo *Beyond consensual motifs: an analysis of DNA curvature within Escherichia coli promoters* de Casa *et al.* (2022), os autores realizaram um estudo sobre as principais características de sequências promotoras, com foco na curvatura dos promotores de *E. coli*. O objetivo do estudo foi definir regiões relevantes para a classificação computacional.

Para a análise realizada, 3895 sequências promotoras de *E. coli*, exibidas na Tabela 11, foram extraídas do *RegulonDB* (Gama-Castro *et al.* (2008)). O comprimento das sequências foi aumentado para 240 pares de base utilizando o genoma de referência. Para as sequências não-promotoras, foram gerados quatro grupos comparativos, sendo que cada um contava com a mesma quantidade de sequências que o conjunto de dados promotores. Os três primeiros, possuíam diferentes proporções de bases nitrogenadas A/T e G/C que se assemelham a partes do genoma da bactéria *E. coli* e o quarto grupo compreendia sequências completamente aleatórias.

Tabela 11 – Sequências de *E. coli* agrupadas pelo fator σ regulador.

Fator σ	**Nº de Sequências**
σ^{24}	521
σ^{28}	140
σ^{32}	329
σ^{38}	211
σ^{54}	94
σ^{70}	1970
Desconhecido	630

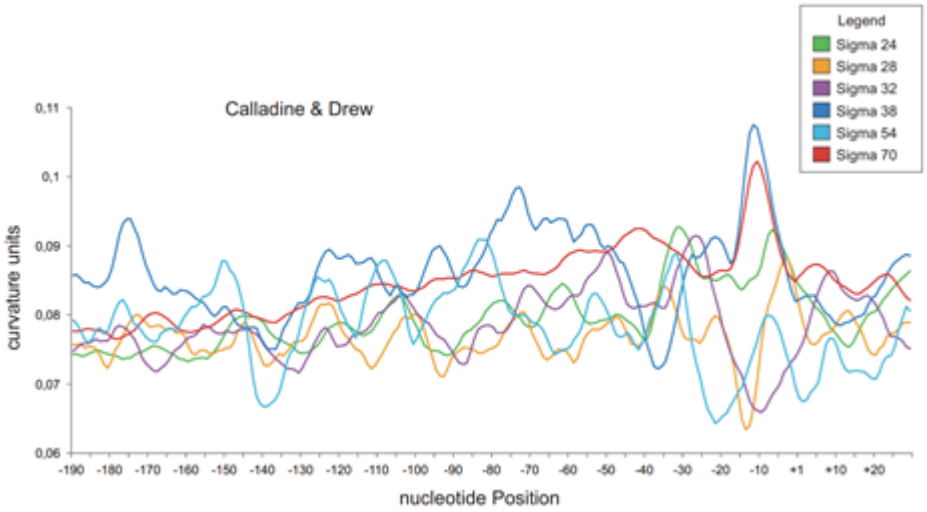
Fonte: Casa *et al.* (2022)

O cálculo da curvatura do DNA foi realizado pelos autores utilizando um *script* desenvolvido na linguagem *Python* disponibilizado por Gohlke (2020) cujo princípio é determinar, primeiramente, a trajetória do DNA para depois calcular sua curvatura. O *script* em questão dispõe de sete modelos diferentes de curvatura e todos foram utilizados na análise realizada. Diante disso, todas as sequências, promotoras ou não, foram convertidas em valores de curvatura para cada nucleotídeo. Para identificar as principais regiões que apresentam indícios distintos de curvatura entre sequências promotoras e não-promotoras foi utilizado o teste de *Kruskal-Wallis* em cada posição de nucleotídeo, esse método é não paramétrico e costuma ser utilizado na comparação de três ou mais grupos. Visando estruturar grupos com sequências correlatas, as regiões que apresentaram diferenças na curvatura foram utilizadas como entrada para uma análise mineração de dados com a técnica de agrupamento *K-means*. Também, foi aplicado o método de árvore de decisão nas sequências já identificadas como promotoras para classificar de forma direta os dados baseado no fator sigma (σ). A validação do método, se deu por meio de matrizes de confusão e das seguintes métricas de classificação: acurácia, precisão, revocação e F1-*Score*.

De acordo com os autores, os valores médios de curvatura do DNA foram estatisticamente mais significativas para sequências promotoras do que para sequências não-promotoras. Apesar disso, foi verificada uma alta variação da curvatura nos agrupamentos, devido ao algoritmo não ter conseguido separar completamente a classe promotora. Porém, observou-se que uma grande parte das sequências promotoras possuem a estrutura mais curva, o que pode ser utilizado como aspecto complementar para promotores eficientes.

Outro ponto avaliado por Casa *et al.* (2022), foi o uso da curvatura do DNA para especificar o fator σ e nesta constatou-se que não existe diferença significativa entre promotores frente às classes de sigma. Os modelos mais assertivos demonstram que sinais de picos e vales se coincidem entre promotores regulados por diferentes fatores sigma, devido ao fato de que composições distintas de nucleotídeos geram estruturas parecidas. Isso resulta em um reconhecimento de promotores de maneira sobreposta, já que mais de um fator sigma pode regular um mesmo promotor. Na Figura 14 é apresentado o resultado obtido ao aplicar o método *Calladine & Drew (DC)*, bastante eficaz na distinção dos fatores sigma. Nela é possível verificar que existe sobreposição em alguns momentos, como visualizado na análise realizada pelos autores.

Figura 14 – Curvaturas da E. *coli* agrupadas pelo fator sigma.



Fonte: Casa *et al.* (2022)

Através do estudo realizado, os autores identificaram que a utilização da curvatura do DNA é bastante eficaz quando aplicadas em sequências reguladas por σ^{38} e σ^{70}, porém ao ser aplicado em cenários com outros fatores sigma apresentou pouca eficácia. Aplicando as métricas classificatórias, observou-se que o método apresenta alta sensibilidade, por classificar corretamente as sequências de σ^{38} e σ^{70}, porém uma baixa especificidade visto que as sequências reguladas pelos outros fatores foram classificadas erroneamente. Dessa forma, entende-se que o aprimoramento dos algoritmos de predição gira em torno de combinar mais de uma característica físico-química da sequência avaliada.

No artigo *DNA structural and physical properties reveal peculiarities in promoter sequences of the bacterium Escherichia coli K-12* de Martinez *et al.* (2021), os autores realizaram um estudo envolvendo os aspectos entalpia (energia interna), estabilidade e empilhamento de pares de base de uma sequência de DNA com foco em codificar as informações genéticas com objetivo de facilitar a classificação de sequências promotoras e não-promotoras.

Os promotores costumam ser mantidos no nível de sequência, porém possuem diversas variações na arquitetura que, juntamente, com inserção e remoção de pares de base e eventuais mutações, podem prejudicar a classificação de promotores quando esta é fundamentada na existência de nucleotídeos específicos em determinadas regiões de uma sequência. Alguns estudos indicam que aspectos relacionados à energia demonstram ser mais eficazes e incomparáveis frente a outros. Diante disso, a proposta dos autores é, através das características mencionadas anteriormente, identificar as principais diferenças que transformam as regiões promotoras em singulares.

A base utilizada na análise foi composta de 3131 sequências promotoras extraídas do *RegulonDB* (Gama-Castro *et al.* (2008)), conforme detalhado na Tabela 12, as sequências avaliadas possuem comprimento de 81 nucleotídeos (-60 a +20), pois trata-se do tamanho padrão

das sequências disponíveis na base e, também, o fato de que promotores costumam ser identificados nessa faixa. Além dessas, foi considerada uma coleção de promotores regulados pelo σ^{54}, aprovados experimentalmente, para gerar uma validação extra ao método estabelecido. A mesma quantidade (3131) de sequências não promotoras foram geradas através de um *script* desenvolvido na linguagem *Python* que mistura as sequências promotoras originais conservando o comprimento (81 nucleotídeos) e o conteúdo de AT. Com o intuito de reconhecer os principais aspectos das sequências promotoras foi realizada a comparação entre as sequências originais e dois tipos de sequências de controle (não-promotoras), sendo elas: (*i*) sequências promotoras embaralhadas (*ii*) sequências codificantes de mesmo tamanho.

Tabela 12 – Sequências promotoras agrupadas pelo fator σ regulador, juntamente com a quantidade de genes associados e o percentual de AT em sua composição.

Fator σ	**Nº de Sequências**	**Genes Associados**	**%AT**
σ^{24}	508	429	59,8
σ^{28}	133	129	57,14
σ^{32}	299	287	54,19
σ^{38}	232	157	56,5
σ^{54}	90	83	59,03
σ^{70}	1869	1456	57,94

Fonte: Adaptado de Martinez *et al.* (2021)

Foi realizada a conversão das características avaliadas em valores estruturais visando a classificação de sequências promotoras e não-promotoras. Para determinar os valores das duplas de nucleotídeos, foi atribuído um valor estrutural a cada posição composta por dois nucleotídeos. Para isso, foram considerados estudos sobre a fusão do DNA sendo que a equação (4.2) denota o cálculo realizado para definir a estabilidade livre dos pares de base, enquanto a (4.3), baseou o cálculo dos valores de entalpia de todos os nucleotídeos visualizados em janelas deslizantes sobrepostas. Já o empilhamento de pares de base foi definido com base na equação (4.4), onde *A* representa a energia de empilhamento, $\sum a$ a soma de todos eles e *n* o número de dinueleotídeos.

$$\mathrm{G} = \Delta G^{0}_{\mathrm{i,\ i+1}} \tag{4.2}$$

$$\mathrm{H} = \Delta H^{0}_{\mathrm{i,\ i+1}} \tag{4.3}$$

$$A = \frac{\sum a}{n} \tag{4.4}$$

Na Tabela 13 são representados os valore utilizado para cada uma das características avaliadas, sendo estes agrupados por cada variação de dinueleotídeos. De acordo com os autores, os valores exibidos precisam ser normalizados devido à diferença de escala identificada e

a necessidade de avaliar as características em paralelo, para isso foi utilizada a equação (4.5), onde *n* representa a normalização; *v* o valor da característica (entalpia, empilhamento de pares de bases ou estabilidade livre) que está sendo normalizada, *min* refere-se ao menor valor e *max* refere-se ao maior valor da característica, conforme exibido na tabela, obtidos para o aspecto avaliado.

$$n = \frac{v - min}{max - min} \tag{4.5}$$

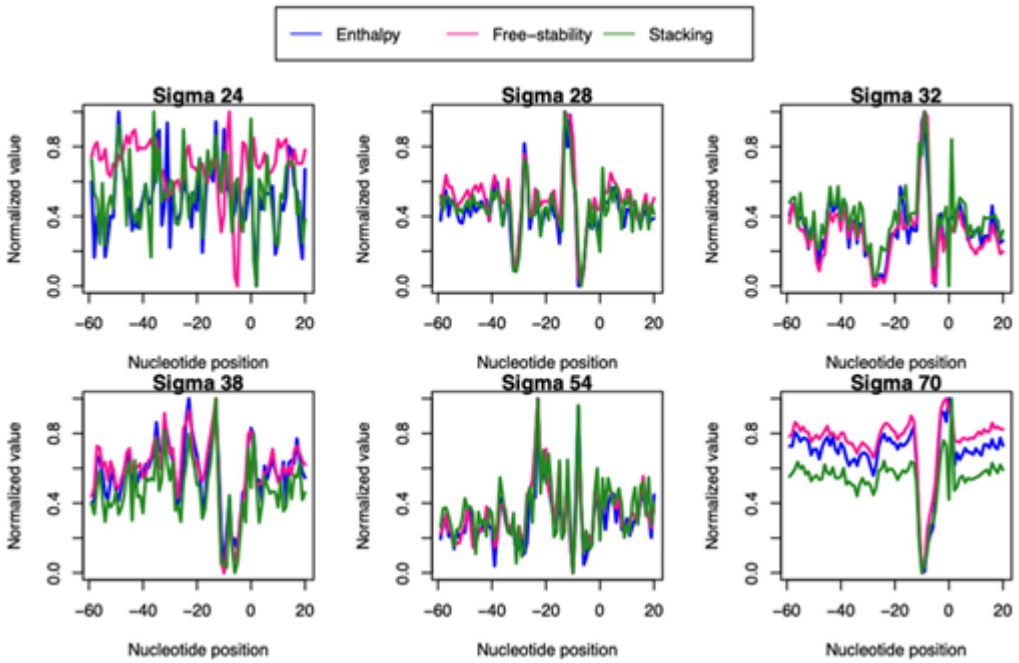
Tabela 13 – Valores utilizados para entalpia, estabilidade livre e empilhamento de pares de bases

Duplex	**Entalpia**	**Estabilidade**	**Empilhamento**
AA/TT	-7,6	-1,00	-5,37
AT/TA	-7,2	-0,88	-6,57
TA/AT	-7,2	-0,58	-6,57
CA/GT	-8.5	-1,45	-6,78
GT/CA	-8,4	-1,44	-10,51
CT/GA	-7,8	-1,28	-6,78
GA/CT	-8,2	-1,30	-9,81
CG/GC	-10,6	-2,17	-14,59
GC/CG	-9,8	-2,24	-9,69
GG/CC	-8,0	-1,84	-8,26

Fonte: Adaptado de Martinez *et al.* (2021)

Através da Figura 15 é observado que, com exceção do regulador σ^{24}, em todos os promotores as características avaliadas apresentaram padrão semelhante de distribuição. Nesses, é possível observar que os aspectos avaliados se sobrepõem, demonstrando que todos são relevantes frente à atividade do promotor. Visando justificar a sobreposição entre as características, os autores utilizaram a correlação, através do teste de *Spearman* devido aos dados analisados não seguirem uma distribuição normal. Através dele, foi possível observar que: (*i*) entre entalpia e empilhamento de pares de base existe uma correlação alta (R = 0,806); (*ii*) entre entalpia e estabilidade livre existe um correlação moderada (R = 0,53); (*iii*) entre estabilidade e empilhamento de pares de base existe uma correlação moderada (R = 0,478). As sequências reguladas pelo σ^{24}, apresentaram ruídos consideráveis em seu perfil de aspectos estruturais avaliados e, diferentemente dos demais fatores, não mostraram sobreposição entre elas.

Figura 15 – Valores médios de Entalpia, Estabilidade Livre e Empilhamento por todos os 80 nucleotídeos, separados pelo fator σ regulador.



Fonte: Martinez *et al.* (2021)

De acordo com Martinez *et al.* (2021), os promotores e não-promotores apresentam diferenças físicas significativas que podem ser utilizadas no desenvolvimento de novos métodos classificadores de sequências. A entalpia, a estabilidade e o empilhamento de pares de bases são fatores que contribuem para essa distinção. Os promotores apresentam maior composição de bases AT e tamanho menor que os não-promotores, o que resulta em menor estabilidade livre.

Avaliando os artigos mencionados foi possível observar que a compreensão dos aspectos de cada tipo de sequência avaliada, pode contribuir para o aprimoramento de estudos e sua aplicação em novas abordagens. Segundo os autores, atributos como: curvatura, flexibilidade, estabilidade, entropia e energia de empilhamento de bases são favoráveis para a classificação de elementos genômicos.

Com isso, Dall'alba (2018) realizou uma análise das principais ferramentas para classificação de sequências de DNA, incluindo o BacPP. A partir dessa análise, o autor propôs um método para determinar um limite entre sequências promotoras e não-promotoras. O método utiliza a codificação dos valores de estabilidade de duplex de DNA para as sequências relacionadas ao σ^{28}. O objetivo é desenvolver um algoritmo para melhorar o desempenho das ferramentas existentes para realizar esse tipo de classificação, trabalhando como uma validação em segunda etapa. A decisão de iniciar o tratamento pelo fator σ^{28} foi fundamentada em pesquisas anteriores, que demonstraram que a codificação baseada em valores de estabilidade apresenta um desempenho comparável ao reportado na literatura para os fatores σ^{28} e σ^{54}. Em uma comparação entre os dois fatores, identificou-se que o σ^{28} possui um conjunto de dados

maior, fazendo com que a análise realizada seja mais confiável.

Para o treinamento do algoritmo, foram utilizados dois conjuntos de dados de sequências de DNA de comprimento regular com 81 nucleotídeos. O primeiro conjunto, com 1306 sequências, corresponde a fatores σ alternativos, todas classificadas como promotoras e obtidas do *RegulonDB* (Gama-Castro *et al.* (2008)). O segundo conjunto, com apenas sequências não-promotoras, foi gerado a partir de um *script* na linguagem *Python*. Já para a codificação em valores de estabilidade, utilizou-se como base o conceito de que a estabilidade do duplex de DNA pode ser obtida através do cálculo de sua energia livre (ΔG), denotado na expressão (4.6).

$$\Delta \mathrm{G} = -(\Delta G^0_{\mathrm{ini}} + \Delta G^0_{\mathrm{sym}}) + \sum_{i=1}^{n-1} \Delta G^0_{\mathrm{i,\,j+1}} \tag{4.6}$$

Os termos $\Delta G^0_{\mathrm{ini}}$ e $\Delta G^0_{\mathrm{sym}}$ são medidas da estabilidade do duplex de DNA que ponderam o equilíbrio de pares de base individuais e a simetria da sequência de DNA. Portanto, para a predição de promotores e não-promotores podem ser desconsiderados. A obtenção dos valores de estabilidade foi realizada por meio de uma janela deslizante sobreposta, ou seja, um nucleotídeo por vez. Isso permitiu obter um valor de estabilidade para cada um deles, considerando também o contexto das bases vizinhas. Isso porque, a estabilidade de um nucleotídeo no duplex de DNA depende não apenas de sua própria identidade, mas também das identidades dos nucleotídeos vizinhos. O processo para a obtenção dos valores de estabilidade foi repetido para cada uma das sequências, sendo elas promotoras ou não.

Os valores das regras foram extraídos da ferramenta BacPP para melhor compreender seu funcionamento e aprimorá-lo. A sequência obtida é conhecida como "protótipo de regra"e representa uma simplificação da aprendizagem de máquina aplicada. No cenário de análise, considerando o fator σ^{28}, obteve-se uma sequência de 80 nucleotídeos, com um valor por posição de nucleotídeo. A análise dos dados obtidos, foi realizada com base em três fases principais.

- **Fase 1**: foi encontrada a diferença entre o valor extraído da regra (Rv) e o valor de estabilidade obtido (Sv), através da expressão (4.7), com essa diferença é possível identificar as posições mais importantes de um nucleotídeo no ponto de vista do algoritmo.

$$V_{dif} = Rv - Sv \tag{4.7}$$

- **Fase 2**: foi fundamentada através da Análise de Componentes Principais (PCA) que consiste em reduzir o tamanho do conjunto de dados avaliado mantendo o máximo possível das informações originais. Na aplicação em questão, foram definidas as regiões da sequência que possuem as principais características que dependem do fator σ. Na Ta-

bela 14, são apresentados três componentes diferentes, cada um formado de uma faixa de nucleotídeos.

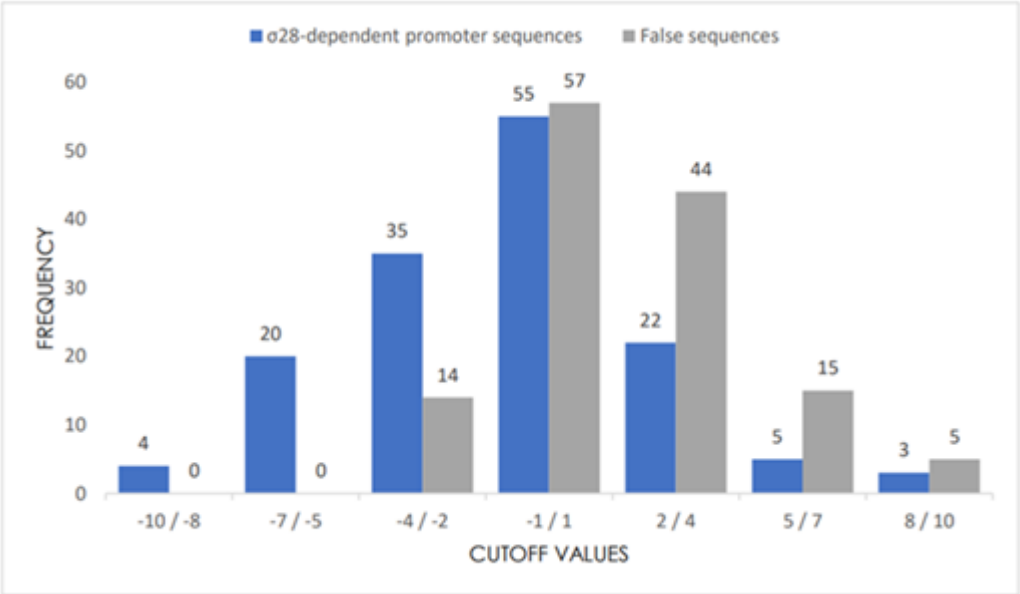
Tabela 14 – Regiões descritas pela PCA

Componente Principal	Faixa Nucleotídeos
1	- 12 até - 16
2	- 46 até - 51 - 41 até - 44 - 31 até - 35 - 8 até - 11 + 9 até + 12
3	- 23 até - 27

Fonte: Dall'alba (2018)

- **Fase 3**: foi realizada a soma de cada nucleotídeo chegando a um valor pra cada sequência analisada relacionadas ao fator σ^{28}, sendo ela do conjunto de promotores ou de não-promotores. Os valores obtidos foram plotados em um histograma para identificar um valor de corte entre as sequências promotoras e não-promotoras. Na Figura 16, é apresentado o histograma resultante, estando em azul os promotores e em cinza os não-promotores.

Figura 16 – Histograma com os valores obtidos.

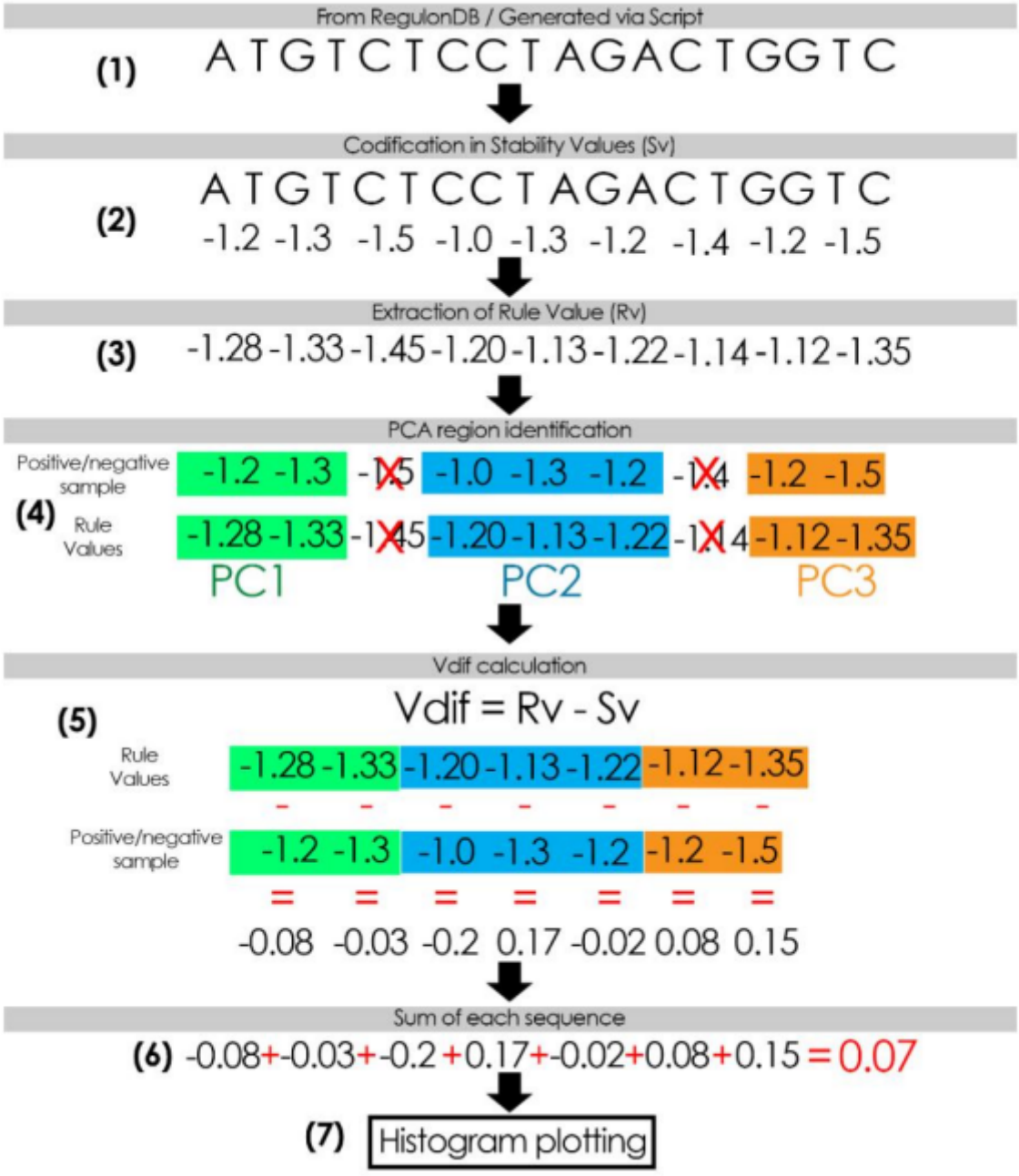


Fonte: Dall'alba (2018)

Segundo o autor, após avaliar os dados obtidos e apresentados no histograma o valor de corte é zero, pois, nesse ponto, o percentual obtido é adequado tanto para sequências promotoras quanto para não-promotoras. Portanto, é possível afirmar que sequências com valor obtido

maior que zero são classificadas como não-promotoras, enquanto sequências com valor obtido menor que zero são classificadas como promotoras. O desenvolvimento realizado por Dall'alba (2018) é diretamente relacionado ao presente trabalho, pois neste o algoritmo será desenvolvido computacionalmente e validado por meio de métricas de classificação. Isso permitirá confirmar se o uso do algoritmo como validador de segunda etapa no BacPP é eficaz. Na Figura 17 está sendo apresentado, graficamente, a metodologia realizada pelo autor para o desenvolvimento do algoritmo.

Figura 17 – Metodologia proposta para aprimoramento da classificação.

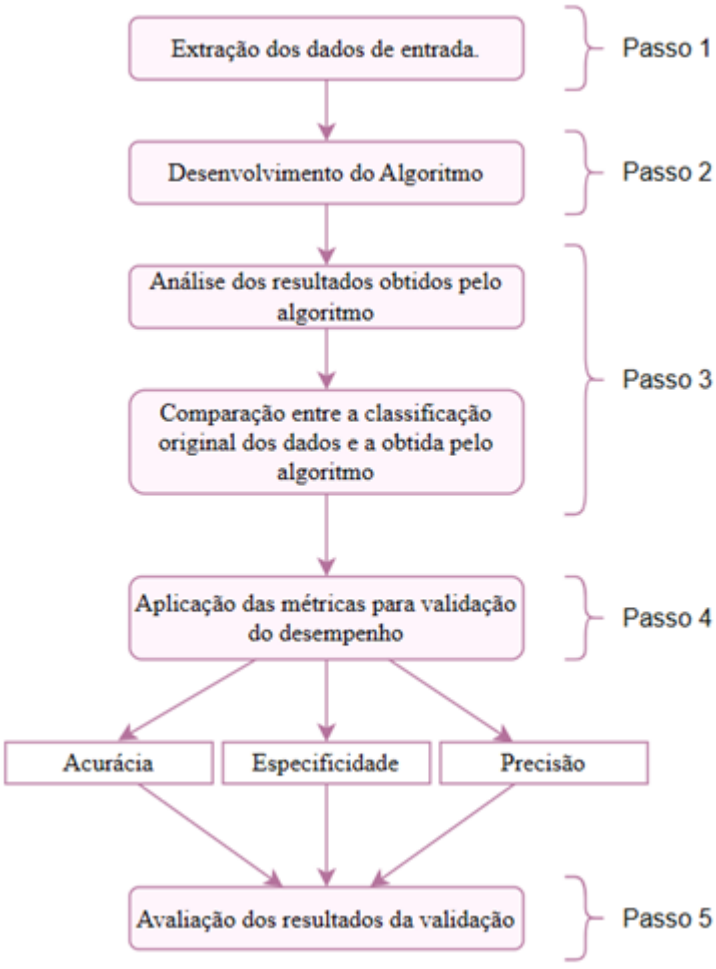


Fonte: Dall'alba (2018)

5 METODOLOGIA

A metodologia que foi utilizada no desenvolvimento desse trabalho pode ser resumida pelo fluxograma apresentado na Figura 18. O primeiro passo foi a extração dos dados previamente classificados e que foram utilizados no treinamento do algoritmo, seguido da implementação do algoritmo idealizado por Dall'alba (2018) e apresentado na Figura 17 da Seção 4 do presente trabalho. O terceiro passo envolve a análise dos resultados obtidos ao executar a implementação proposta, sendo estes analisados e comparados com a classificação original de cada sequência no quarto passo. Na etapa seguinte foram aplicadas as métricas para validação de desempenho do algoritmo e, por fim, os resultados obtidos por essa validação foram analisados para verificar se o método implementado foi eficaz ou não.

Figura 18 – Metodologia proposta para aprimoramento da classificação.



Fonte: O autor (2024).

5.1 BASE DE DADOS

O conjunto de sequências promotoras utilizado no treinamento e validação do algoritmo foi extraído da base de dados RegulonDB [1]. Apenas sequências reguladas pelo fator σ^{28} da bactéria *E. coli* foram utilizadas na validação, uma vez que o algoritmo foi desenvolvido com base nesse modelo específico. Consequentemente, o uso de sequências reguladas por outros fatores ou provenientes de outras bactérias pode resultar em distorções nos resultados. Como exemplo negativo, foi gerado o mesmo número de sequências aleatórias, através de um *script* desenvolvido na linguagem *Python*, sendo utilizadas como sequências não-promotoras e assim garantir uma maior assertividade nos resultados obtidos. Cada sequência contém, no mínimo, 81 nucleotídeos, sendo que cada par foi convertido em um valor conforme a regra estabelecida por Silva *et al.* (2014), baseada nas regras de aprendizado de máquina do BacPP.

5.2 METODOLOGIA DE DESENVOLVIMENTO DE *SOFTWARE*

A implementação do algoritmo foi realizada na linguagem de programação *Python*, ela serviu como uma validação em segunda etapa da ferramenta BacPP, por isso a importância de manter a mesma linguagem e, como mencionado na Seção 3 desse trabalho, o BacPP foi totalmente desenvolvido em Python. Além disso, a linguagem é uma das mais populares da atualidade e dentre as características principais está o fato de apresentar uma curva de aprendizagem baixa. Também, é amplamente utilizada na ciência de dados, principalmente, por conta de suas bibliotecas específicas para a aplicação (NETTO; MACIEL, 2021). Para o desenvolvimento, será utilizada a *IDE Visual Studio Code*, devido ao fato de ser altamente extensível contando com extensões que auxiliam no desenvolvimento.

O processo de implementação foi baseado na metodologia ágil *Scrum*, que se refere a um processo que contempla as atividades de: requisitos, projeto, evolução e entrega, com o intuito de chegar ao resultado desejado de forma rápida e segura. Cada atividade mencionada possui tarefas que precisam ser executadas dentro de um período de tempo específico conhecido como *sprint*. O tempo de duração varia de acordo com a complexidade e o tamanho da implementação a ser feita, essas características, também, influenciam na quantidade de *sprints* a ser executada, isso porque, em um cenário no qual o projeto final é grande, seu desenvolvimento costuma ser dividido em entregas menores, ou seja, uma aplicação final pode estar vinculada a mais do que uma *sprint* (PRESSMAN; MAXIM, 2021). Além disso, o projeto foi acompanhado integralmente pelo *stakeholder*, ou seja, pelas partes interessadas no projeto, que desempenharam um papel essencial no apoio ao processo como um todo, garantindo que o desenvolvimento fosse viabilizado de forma para garantir a entrega final com término e dentro do prazo previsto (PRESSMAN; MAXIM, 2012).

[1] https://regulondb.ccg.unam.mx/

O desenvolvimento de *software* visa resolver algum tipo de problema e, por isso, é extremamente importante entender as necessidades que precisam ser atendidas por ele, tanto na camada "negócios"como na técnica. Projetos executados sem esse estudo inicial costumam apresentar diversos aspectos negativos como: entregas de baixa qualidade, retrabalho, entregas atrasadas, etc. (PRESSMAN; MAXIM, 2021). O levantamento de requisitos conta com metodologias que orientam o processo tornando-o mais assertivo, o método Ciclo *Plan - Do - Check - Act* (PDCA) é bastante utilizado, pois auxilia, gerencialmente, nas tomadas de decisões, visando maior assertividade nos prazos e cumprimento de metas. Em paralelo a ele é comum ser utilizada a metodologia chamada 5W2H que simboliza as palavras em inglês: *What* (O quê), *Who* (Quem), *Why* (Por quê), *When* (Quando), *How* (Como) e *How Much* (Quanto Custa). Cada termo, referencia perguntas básicas que abordam questões fundamentais para o desenvolvimento de *software* (PRESSMAN; MAXIM, 2012).

No início do processo, é comum levantar os requisitos principais. Após a entrega da primeira versão, seja em ambiente controlado ou para o cliente final, podem ser levantados novos requisitos, que serão incorporados em uma segunda etapa do desenvolvimento. Isso ocorre devido a duas realidades: (*i*) é impossível que os envolvidos descrevam um sistema completo antes de observarem o *software* operacional; (*ii*) é difícil que os envolvidos descrevam os requisitos de qualidade necessários para o *software* antes de vê-lo em ação. Dessa forma, o uso da metodologia *Scrum* auxiliou nesse cenário, devido à possibilidade de atender os novos requisitos levantados, por meio de seu princípio de *sprints* (PRESSMAN; MAXIM, 2021).

Após a conclusão do levantamento de requisitos, foi realizada uma avaliação técnica para definir o que deve ser executado e como isso deve ser feito. Nessa etapa, definiu-se as bibliotecas, comandos, funções, etc. que foram utilizadas para atender ao objetivo final. O desenvolvimento do algoritmo foi realizado em partes, sendo que cada uma passou por três passos fundamentais: (*i*) desenvolvimento do código; (*ii*) testes unitários; (*iii*) entrega e validação junto ao *Stakeholder*.

5.3 VALIDAÇÃO DOS RESULTADOS

Para a validação do algoritmo desenvolvido foram utilizadas três métricas distintas, dessa forma, foi possível validar o desempenho do mesmo em aspectos distintos. A acurácia (expressão 2.8 da seção 2) permitiu avaliar, de forma geral, a assertividade da classificação realizada pelo algoritmo, ou seja ela retornou o percentual de predições corretas, para sequências promotoras e não-promotoras. Com a especificidade (expressão 2.11 da seção 2) foi possível medir a classificação correta de amostras negativas, ou seja, sequências não-promotoras. Já a precisão (expressão 2.9 da seção 2) possibilitou a validação do desempenho do algoritmo frente às amostras positivas. Para o método ser considerado eficaz é importante apresentar índice alto de acurácia, especificidade e precisão. Dentre todas as métricas possíveis para problemas de

classificação, estas foram escolhidas por proporcionarem uma avaliação equilibrada e completa do desempenho do algoritmo.

5.4 IMPLEMENTAÇÃO

A aplicação da metodologia descrita anteriormente ocorreu na segunda fase deste trabalho, na qual os conhecimentos adquiridos até então foram utilizados conforme o cronograma previsto na Tabela 15. Inicialmente, foram realizados o levantamento dos requisitos, as modelagens e as especificações necessárias, que foram apresentados ao *stakeholder* para avaliação e sugestão de possíveis ajustes antes do início do desenvolvimento. Após a aprovação e a formalização das definições iniciais em nível técnico, deu-se início ao desenvolvimento.

Na primeira etapa, foi realizada a implementação da leitura da sequência e da conversão dos caracteres em seus respectivos valores numéricos. Em seguida, na segunda etapa, foram desenvolvidos os cálculos para a classificação da sequência de DNA como promotora ou não-promotora. Após a conclusão e os testes individuais de cada etapa, foi realizado um teste final integrando todo o processo, permitindo a avaliação completa dos resultados obtidos e a aplicação das métricas de validação.

Tabela 15 – Cronograma previsto.

Tarefa	**Estimativa**
Levantamento dos requisitos	Mês 1
Especificação técnica	Mês 1
Apresentação das definições ao *Stakeholder*	Mês 1
Desenvolvimento Entrega 1	Mês 2
Testes Entrega 1	Mês 2
Entrega Entrega 1	Mês 2
Desenvolvimento Entrega 2	Mês 3
Testes Entrega 2	Mês 3
Entrega Entrega 2	Mês 3
Testes finais	Mês 4
Aplicação e análise das métricas de validação	Mês 4
Escrita dos resultados obtidos na monografia	Mês 5
Entrega da monografia para avaliação do orientador	Mês 5
Realização dos ajustes apontados pelo orientador	Mês 5
Entrega da monografia revisada	Mês 5
Apresentação do trabalho para a banca avaliadora	Mês 6

6 RESULTADOS

6.1 APLICAÇÃO PRÁTICA DO ALGORITMO

O processo de desenvolvimento da implementação foi conduzido com base nos princípios da metodologia ágil *Scrum*, amplamente utilizado na engenharia de software. Antes de iniciar o desenvolvimento do algoritmo o *stakeholder* apresentou os principais requisitos funcionais que o sistema deveria atender, organizados em etapas sequenciais: *(i)* entrada e validação dos dados; *(ii)* conversão dos nucleotídeos em seus respectivos valores numéricos; *(iii)* localização dos componentes principais da sequência; *(iv)* comparação com os valores definidos pelas regras; *(v)* cálculo das diferenças entre as sequências avaliadas e as regras de referência; e *(vi)* soma final dos valores processados.

O planejamento e a execução das atividades foram organizados em ciclos semanais, com reuniões regulares entre o desenvolvedor e o *stakeholder*. Durante essas reuniões, foram discutidos o progresso das tarefas, possíveis ajustes e validações necessárias, assegurando que o desenvolvimento permanecesse alinhado às expectativas e aos requisitos previamente definidos.

Durante o desenvolvimento, cada etapa foi submetida a testes unitários específicos, com o objetivo de identificar e corrigir eventuais erros de forma ágil. Esses testes permitiram validar individualmente cada funcionalidade criada, garantindo o pleno funcionamento do código antes de avançar para as etapas subsequentes. Esse processo iterativo e incremental possibilitou não apenas a mitigação de riscos, mas também a entrega de um algoritmo alinhado às especificações propostas.

A implementação do algoritmo foi baseada na Figura 17, e algumas funções específicas foram criadas para obter o resultado desejado. Essas funções serão mencionadas ao longo da seção atual. No entanto, antes de aplicá-las, foi necessário considerar elementos fundamentais, como a leitura e a tratativa da sequência. O código completo está disponível no Anexo A.

Para isso, em parte do código, optou-se por utilizar a biblioteca consolidada *Biopython*, amplamente utilizada na comunidade de bioinformática. *Biopython* é uma biblioteca de código aberto desenvolvida em *Python*, que fornece ferramentas para diversas tarefas em bioinformática e biologia computacional. Entre seus módulos, destaca-se o módulo Bio, utilizado principalmente para o tratamento e interpretação de sequências de DNA, RNA e proteínas.

O BacPP recebe sequências no formato *Fasta*, seja por digitação do usuário ou por importação de arquivo, o que exige que a entrada do algoritmo esteja preparada para ambos os formatos. Para solucionar esse ponto, foi utilizado o método *SeqIO* disponível no módulo Bio da biblioteca *Biopython*, que permite ler e escrever arquivos de sequência em diferentes formatos. Esse processo de leitura armazena os dados em listas apenas se a sequência for considerada

válida. Para isso, a sequência deve atender a dois requisitos: ter pelo menos 81 posições e conter apenas as letras válidas (A, T, C, G), independentemente do formato das letras.

Na Figura 19, é apresentado um exemplo do formato aceito e tratado na entrada. Ou seja, se a entrada for diferente do modelo não será lida corretamente e retornará que a entrada é inválida.

Figura 19 – Exemplo de arquivo aceito como entrada pelo algoritmo.

```
>Example 1
GAATTGTGCTGTTACGATGTCGCCTCTTCAATTATGTTATCGTGTCTTGCACCCAATGTTATGTTAATCCTCAATGATTTC
```

Fonte: O autor (2024).

Um arquivo *Fasta* pode conter uma ou várias sequências, e cada sequência deve possuir pelo menos 81 letras. Em casos em que uma sequência excede essa quantidade mínima, é necessário realizar uma derivação para dividi-la em fragmentos menores, prontos para análise individual. Esse processo permite que várias subsequências de uma sequência maior sejam verificadas individualmente quanto ao seu potencial de atividade promotora, garantindo que o algoritmo foque em segmentos específicos e reduza a chance de interpretações equivocadas. Na Figura 20, esse processo de derivação é exemplificado, mostrando que uma sequência com 85 nucleotídeos gera 5 derivações. Isso significa que, a partir dessa sequência original, 5 novas subsequências serão avaliadas individualmente, sendo possível que apenas algumas delas sejam classificadas como promotoras.

Figura 20 – Derivação de uma sequência.

```
Sequência Original:
    ATGCGTACGTAGCTAGCTAGCGTACGTAGCTAGGCTAGCTAGCTACGTAGCCTGATCGGATCGTAGCTAGCGTAGCTAGGATCGT

Derivações:
    1) ATGCGTACGTAGCTAGCTAGCGTACGTAGCTAGGCTAGCTAGCTACGTAGCCTGATCGGATCGTAGCTAGCGTAGCTAGG
    2) TGCGTACGTAGCTAGCTAGCGTACGTAGCTAGGCTAGCTAGCTACGTAGCCTGATCGGATCGTAGCTAGCGTAGCTAGGA
    3) GCGTACGTAGCTAGCTAGCGTACGTAGCTAGGCTAGCTAGCTACGTAGCCTGATCGGATCGTAGCTAGCGTAGCTAGGAT
    4) CGTACGTAGCTAGCTAGCGTACGTAGCTAGGCTAGCTAGCTACGTAGCCTGATCGGATCGTAGCTAGCGTAGCTAGGATC
    5) GTACGTAGCTAGCTAGCGTACGTAGCTAGGCTAGCTAGCTACGTAGCCTGATCGGATCGTAGCTAGCGTAGCTAGGATCG
```

Fonte: O autor (2024).

Para suportar essa flexibilidade na manipulação de dados, utilizou-se o conceito de listas, que permite armazenar e acessar as subsequências derivadas de maneira dinâmica e organizada. Desde a entrada dos dados, as sequências são armazenadas em listas, o que facilita tanto a derivação quanto o armazenamento dos resultados intermediários e finais. O uso de listas permite que cada sequência, ou subsequência derivada, seja facilmente acessada e manipulada conforme necessário, simplificando operações como a verificação de regiões específicas.

Embora listas possam apresentar uma desvantagem em termos de velocidade quando o volume de dados é muito grande, a escolha por utilizá-las se justifica pelo foco principal do

algoritmo: flexibilidade na manipulação das sequências e facilidade de acesso a partes específicas, em vez de desempenho em termos de velocidade. A estrutura de listas permite iterar sobre cada subsequência e realizar operações específicas, como checar se uma sequência derivada atende aos critérios de promotor, de forma eficiente e direta. Assim, optou-se por utilizar listas em todo o processo, maximizando a flexibilidade e o controle sobre as sequências e permitindo uma adaptação rápida a modificações nos critérios de análise, que são características fundamentais para a manipulação de dados biológicos.

A primeira etapa do algoritmo proposto consiste em converter cada par de nucleotídeos em seu respectivo valor numérico, com base na estabilidade de cada par. Os valores atribuídos para essa conversão estão disponíveis na Tabela 16, que fornece a correspondência entre pares de nucleotídeos e seus valores numéricos associados.

Tabela 16 – Valor de estabilidade para cada par de nucleotídeo

Par de Nucleotídeo	**Valor**
AA	-1.00
AT	-0.88
TA	-0.58
AG	-1.30
GA	-1.30
TT	-1.00
AC	-1.45
CA	-1.45
TG	-1.44
GT	-1.44
TC	-1.28
CT	-1.28
CC	-1.84
CG	-2.17
GC	-2.24
GG	-1.84

Na implementação, utilizou-se o conceito de dicionários para armazenar cada par de nucleotídeo e seu valor correspondente, pois essa estrutura permite acessar rapidamente os valores diretamente por sua chave, que neste caso é o par de nucleotídeos, tornando a consulta rápida e eficiente. Além disso, o dicionário contribui para a legibilidade do código, eliminando a necessidade de instruções condicionais complexas para associar cada par ao seu valor. Essa estrutura facilita a manutenção, pois novos pares e valores podem ser adicionados ou alterados facilmente, mantendo a organização e a clareza das informações.

A função *converteNucleotideo*, exibida na Figura 21, é responsável por converter pares de nucleotídeos em valores numéricos, conforme uma regra de conversão específica. Ela recebe como entrada uma lista de sequências e retorna uma estrutura de listas aninhadas contendo os valores convertidos. Cada par de nucleotídeos é avaliado individualmente, e seu valor é atribuído de acordo com o dicionário mencionado anteriormente.

Essa estrutura hierárquica de listas simplifica a manipulação e organização das informações, proporcionando uma padronização importante para as etapas subsequentes do algoritmo. Além disso, essa abordagem torna o código mais modular e escalável, permitindo adaptações para diferentes regras de conversão, caso necessário. Após a execução da função, os processos subsequentes operam diretamente com as listas convertidas, simplificando o fluxo de análise e reduzindo a necessidade de processamento redundante nas etapas seguintes.

Figura 21 – Implementação de função que converte o par de nucleotídeo em valor de estabilidade

```
def converteNucleotideo(listaSequencias):
    listaConvertida = []

    for sequencia_i in listaSequencias:
        convSublista = []

        for string in sequencia_i:
            tamanho = len(string)
            sequenciaConvertida = []

            for pos in range(tamanho - 1):
                par = string[pos:pos + 2]
                if par in regraConversao:
                    resultadoAux = regraConversao[par]

                sequenciaConvertida.append(resultadoAux)

            convSublista.append(sequenciaConvertida)

        listaConvertida.append(convSublista)

    return listaConvertida
```

Fonte: O autor (2024).

A função *componentePrincipal*, exibida na Figura 22, é responsável por definir os componentes principais, aplicando valores específicos a determinadas posições dentro de uma sequência, conforme regras estabelecidas a partir da extração de dados de uma rede neural desenvolvida por Silva *et al.* (2014). Esses componentes principais recebem valores numéricos baseados na estabilidade dos nucleotídeos em posições específicas, enquanto as demais posições, que estão fora dessas regiões principais, recebem o valor zero e são desconsideradas no processo de predição.

Para realizar essa tarefa, a função avalia cada posição na sequência e identifica se ela corresponde a um dos componentes principais (definidos como PC1, PC2 ou PC3). Quando uma posição corresponde a um componente principal, um valor numérico específico é aplicado, de acordo com as regras disponíveis na Tabela 14. Com essa estrutura, o algoritmo é capaz de executar uma análise detalhada e focada, priorizando as posições críticas para a classificação e minimizando a influência de regiões irrelevantes.

Figura 22 – Implementação de função que identifica os componentes principais da sequência

```
def componentePrincipal(item, pos):
    # PC1
    if pos >= 46 and pos <= 50:
        resultado = item - (-1.67)
    # PC2
    elif ((pos >= 11 and pos <= 16) or (pos >= 18 and pos <= 21) or
          (pos >= 27 and pos <= 31) or (pos >= 51 and pos <= 54) or
          (pos >= 71 and pos <= 74)):
        resultado = item - \
            (((-1.42) + (-1.24) + (-1.16) + (-1.05) + (-1.36)) / 5)
    # PC3
    elif (pos >= 35 and pos <= 39):
        resultado = item - (-1.42)
    else:
        resultado = 0

    return resultado
```

Fonte: O autor (2024).

A função *calculaDiferenca*, exibida na Figura 23, é responsável por comparar a sequência já convertida com os valores das regras citadas na Tabela 14, considerando apenas as regiões que fazem parte dos componentes principais. Ela recebe como entrada uma lista de sequências convertidas e, para cada uma delas, calcula a diferença de valores em relação aos componentes principais, retornando uma estrutura de listas aninhadas que contém as diferenças obtidas.

Sua implementação consiste em iterar sobre cada sublista contida na lista de sequências convertidas. Para cada item da sublista, a função *componentePrincipal* é aplicada, armazenando os valores resultantes em uma lista interna. Esse processo é repetido para todas as sequências, garantindo que cada uma passe pela avaliação nas regiões importantes, enquanto as áreas irrelevantes são ignoradas.

O resultado obtido em cada lista interna é, então, armazenado em uma lista final, que mantém todas as comparações realizadas para as sequências convertidas. A estrutura hierárquica e modular da função facilita a manutenção do código e permite uma análise detalhada das diferenças nas regiões específicas, assegurando que nenhuma sequência relevante seja omitida na análise. Essa abordagem também contribui para a flexibilidade do código, permitindo ajustes e adaptações futuras conforme novas regras ou requisitos sejam incorporados ao processo de análise.

Figura 23 – Implementação de função que calcula a diferença entre a regra e a sequência já convertida

```
def calculaDiferenca(sequenciaConvertida):
    listaDiferenca = []
    for sequenciaConvertida_i in sequenciaConvertida:
        diferencaInterna = []
        for item in sequenciaConvertida_i:
            listaDiferencaCalculada = []
            for componente, indice in enumerate(item, start = 1):
                listaDiferencaCalculada.append(
                    componentePrincipal(indice, componente))
            diferencaInterna.append(listaDiferencaCalculada)
        listaDiferenca.append(diferencaInterna)

    return listaDiferenca
```

Fonte: O autor (2024).

Por fim, a função *soma*, exibida na Figura 24, calcula a soma de cada item das sequências que passaram pelos processos de conversão e análise anteriores. A partir do resultado final dessa função, é possível determinar se uma sequência possui atividade promotora ou não. O critério de classificação considera que uma sequência é promotora se o resultado for menor do que zero, e não promotora se o resultado for maior ou igual a zero.

Em sua implementação, a função começa iterando sobre cada sequência processada. Para cada conjunto de itens, uma variável de soma é inicializada em zero e acumula o valor de cada item da sublista avaliada, com o resultado final arredondado para duas casas decimais. Esse processo é executado para todas as sequências na lista processada. Ao final, a função retorna uma lista contendo os resultados obtidos para cada sequência.

Ela desempenha um papel essencial ao condensar as informações derivadas das etapas anteriores em um único valor numérico, facilitando a interpretação dos resultados. Esse processo simplifica a análise final, permitindo que a decisão sobre a classificação da sequência seja feita de maneira direta e rápida, com base em um único ponto de corte, o valor zero.

Figura 24 – Implementação de função que soma o valor de cada posição da sequência e retorna o resultado

```
def soma(sequencia):
    sequenciaResultado = []
    for sequencia_i in sequencia:
        resultadoAgrupado = []
        for valor in sequencia_i:
            resultado = 0
            for i in valor:
                resultado = resultado + i
            resultadoAgrupado.append(round (resultado,2))
        sequenciaResultado.append(resultadoAgrupado)

    return sequenciaResultado
```

Fonte: O autor (2024).

Além disso, foi desenvolvida uma função para geração de *logs* em cada etapa do processo, bem como um arquivo final em formato CSV que contém as informações da sequência e o resultado final da classificação (promotor ou não promotor). Esses *logs*, embora não influenciem diretamente no resultado da classificação, são fundamentais para o rastreamento detalhado do fluxo de execução do algoritmo, permitindo que cada etapa seja verificada e compreendida. Durante o desenvolvimento, os *logs* foram amplamente utilizados para identificar e corrigir erros, possibilitando ajustes finos em partes específicas do código e facilitando a depuração em caso de resultados inesperados.

Para validar a implementação e assegurar que o algoritmo estivesse funcionando de acordo com o proposto na Figura 17, foram realizados testes manuais (testes de mesa), que analisaram cada etapa e valor intermediário do processamento, permitindo uma verificação completa da lógica implementada. Além disso, os resultados foram comparados com os dados obtidos a partir do mesmo conjunto de sequências reguladas pelo fator σ^{28} utilizado por Dall'alba (2018). Essa comparação focou nas etapas de cálculo das diferenças e da soma final, permitindo uma análise precisa de como o algoritmo se comportava em relação ao esperado. Os resultados obtidos foram satisfatórios, demonstrando a conformidade do algoritmo com o modelo teórico e confirmando sua correta implementação.

Após assegurar a precisão e o funcionamento adequado do algoritmo, iniciou-se a fase de validação tanto do algoritmo de forma isolada quanto em seu papel como validador de segunda etapa.

O BacPP é uma ferramenta modular, e o BacPP-E28 foi desenvolvido com essa mesma estrutura em mente, permitindo que ambos os módulos sejam mantidos de forma independente. Dessa forma, é possível realizar manutenções ou ajustes específicos em um dos módulos sem interferir diretamente no outro, o que aumenta a flexibilidade e facilita a evolução de cada parte do sistema conforme novos requisitos ou dados estejam disponíveis. Embora essa modularidade introduza um pequeno impacto no desempenho devido ao custo computacional adicional, o desempenho não é um requisito prioritário para essa ferramenta. Portanto, esse formato modular não causa impactos negativos no cumprimento de seu objetivo principal, que é a precisão e adaptabilidade na análise de promotores.

6.2 DESEMPENHO ISOLADO DO ALGORITMO

Para avaliar o desempenho do algoritmo, utilizou-se a base de dados RegulonDB Versão 13[1]. Nessa seleção, foram excluídas as sequências sem fator regulador definido, assim como a única amostra associada ao fator σ^{19}. Para a criação do conjunto de dados de sequências falsas, foi desenvolvido um *script* próprio que gerou, de maneira aleatória, a mesma quantidade de sequências reguladas pelo fator σ^{28}, preservando a proporção de bases característica do promo-

[1] https://regulondb.ccg.unam.mx/datasets/browser/RDBECOLIDLF00011

tor alvo da análise. Os resultados dessa análise, juntamente com a quantidade de sequências utilizadas para cada conjunto, estão apresentados na Tabela 17.

Tabela 17 – Desempenho BacPP-E28 utilizando base RegulonDB

Fator	**Qtde Sequências**	**Qtde Acertos**	**% Acertos**
Falsos	146	99	67,81
σ^{28}	146	92	63,01
σ^{24}	522	262	50,19
σ^{32}	311	111	35,69
σ^{38}	239	135	56,49
σ^{54}	99	32	32,32
σ^{70}	1997	1369	68,65

Foi realizada uma comparação entre as sequências reguladas pelo fator σ^{28} e cada um dos demais conjuntos mencionados, gerando a matriz de confusão apresentada na Tabela 18.

A proporção de valores na diagonal principal, que representa as predições corretas, em relação à diagonal secundária, que representa as predições incorretas, indica o nível de precisão geral do modelo. Valores elevados na diagonal principal sugerem uma alta taxa de acertos, enquanto valores elevados na diagonal secundária destacam áreas que podem ser aprimoradas, fornecendo pontos de melhorias valiosos para futuras adaptações. Falsos positivos podem levar a interpretações equivocadas, enquanto falsos negativos podem resultar na perda de sequências biologicamente relevantes, tornando a análise destes fundamental.

Nesse procedimento, as sequências reguladas pelo fator σ^{28} foram consideradas como exemplos verdadeiros, enquanto os exemplos falsos foram representados tanto por sequências falsas quanto por sequências reguladas por outros fatores σ. Essa abordagem foi adotada para permitir uma avaliação abrangente do desempenho do algoritmo, observando a precisão tanto das classificações corretas quanto das incorretas.

Tabela 18 – Matriz de confusão obtida para cada cenário validado

		Classificação Obtida											
		Falsos X σ^{28}		σ^{24} **X** σ^{28}		σ^{32} **X** σ^{28}		σ^{38} **X** σ^{28}		σ^{54} **X** σ^{28}		σ^{70} **X** σ^{28}	
		+	**-**	**+**	**-**	**+**	**-**	**+**	**-**	**+**	**-**	**+**	**-**
Classificação	+	92	54	92	54	92	54	92	54	92	54	92	54
Verdadeira	-	47	99	260	262	201	111	104	135	67	32	628	1369

O foco dessa avaliação, assim como das demais realizadas, está na comparação entre as sequências falsas e as reguladas pelo fator σ^{28}. Esse enfoque busca compreender o comportamento do algoritmo em relação às sequências promotoras reguladas por esse fator em específico, bem como em relação às sequências que oficialmente não possuem função promotora. A comparação com as sequências reguladas por outros fatores σ é essencial para justificar a exclusividade do algoritmo como validador de segunda etapa para o fator σ^{28}, uma vez que ele demonstra baixa eficácia ao lidar com sequências reguladas por fatores fora de seu escopo.

Após a construção da matriz de confusão, foram aplicadas três métricas de validação - Acurácia, Precisão e Especificidade - para avaliar o desempenho do algoritmo em cada comparação, considerando as duas categorias disponíveis: promotores e não promotores. O objetivo foi verificar a eficácia do algoritmo na classificação correta de exemplos positivos e negativos, com os resultados de cada comparação apresentados na Tabela 19.

Tabela 19 – Desempenho obtido para cada uma das validações realizadas

Validação	**Acurácia (%)**	**Precisão (%)**	**Especificidade (%)**
σ^{28} X Falsos	65,41	66,19	67,81
σ^{28} X σ^{24}	52,99	26,14	50,19
σ^{28} X σ^{32}	44,32	31,40	35,58
σ^{28} X σ^{38}	58,96	46,94	56,49
σ^{28} X σ^{54}	50,61	57,86	32,32
σ^{28} X σ^{70}	68,18	12,78	68,55

Observa-se um equilíbrio significativo no desempenho obtido na comparação entre as sequências falsas e as reguladas pelo fator σ^{28}, com destaque para as métricas de Precisão e Especificidade, que apresentaram valores equilibrados. Essa congruência indica um bom desempenho do algoritmo: a Precisão reflete a capacidade do modelo de classificar corretamente as previsões positivas, enquanto a Especificidade mede sua habilidade de identificar corretamente as previsões negativas. A Acurácia, por sua vez, fornece uma visão geral do desempenho do algoritmo, indicando a proporção de classificações corretas em relação ao total avaliado.

A semelhança entre as três métricas evidencia o bom desempenho do algoritmo, pois é preferível que seja mantido um equilíbrio moderado entre elas, em vez de obter um valor elevado em apenas uma, o que poderia comprometer a robustez da classificação. Essa proporção também sugere que a abordagem do algoritmo é flexível, podendo fornecer resultados confiáveis em um ambiente de pesquisa, onde tanto a assertividade das predições positivas quanto a redução de falsos negativos são essenciais.

6.3 EFICIÊNCIA DO ALGORITMO COMO VALIDADOR DE SEGUNDA ETAPA

Para avaliar a eficiência do BacPP-E28 como validador de segunda etapa, foram selecionadas, de maneira aleatória, 10 sequências reguladas por cada fator σ, além de 10 exemplos negativos, todos retirados dos dados originalmente utilizados para a validação isolada do algoritmo. A seleção dessas sequências foi realizada por meio de um embaralhamento das sequências disponíveis, a partir do qual foram extraídas 10 amostras para cada conjunto. Nessa validação, não foram utilizadas todas as amostras da análise anterior, pois o objetivo era demonstrar, de forma prática, a importância do validador de segunda etapa, bem como justificar sua restrição de uso ao fator σ^{28}. Para esse propósito, um número reduzido de amostras foi suficiente.

Inicialmente o algoritmo BacPP foi executado, configurado especificamente para o fator σ^{28}, sobre cada uma das sequências definidas. Esse procedimento teve como objetivo avaliar os resultados fornecidos pela ferramenta em relação a esse fator regulador, verificando se as sequências analisadas correspondem, de fato, a promotores regulados pelo fator σ^{28}. Em seguida, o algoritmo BacPP-E28 foi aplicado ao mesmo conjunto de dados. O objetivo desta análise foi comparar o desempenho do algoritmo desenvolvido neste trabalho com a ferramenta original na qual ele foi baseado, buscando identificar pontos de convergência e divergência nos resultados obtidos.

A Tabela 20 apresenta os resultados obtidos na comparação entre os algoritmos para as 10 sequências negativas, na qual observa-se que não houve discordâncias entre os resultados. Idealmente, esperava-se que no Exemplo 7 o BacPP-E28 retornasse a classificação "Não Promotor", no entanto, como os exemplos negativos são gerados por meio de um *script* que utiliza probabilidades semelhantes às de um promotor, é provável que a sequência em questão se assemelhe significativamente a uma sequência promotora. Esse fator pode ter contribuído para que o BacPP também a classificasse como promotora.

Tabela 20 – Comparação entre o BacPP e BacPP-E28 - Sequências Falsas

Fator	**Sequência**	**Resultado Esperado**	**Resultado BacPP-28**	**Resultado BacPP-E28**
Falsos	Exemplo 1	Não Promotor	Não Promotor	Não Promotor
Falsos	Exemplo 2	Não Promotor	Não Promotor	Não Promotor
Falsos	Exemplo 3	Não Promotor	Não Promotor	Não Promotor
Falsos	Exemplo 4	Não Promotor	Não Promotor	Não Promotor
Falsos	Exemplo 5	Não Promotor	Não Promotor	Não Promotor
Falsos	Exemplo 6	Não Promotor	Não Promotor	Não Promotor

Continua na próxima página

Fator	Sequência	Resultado Esperado	Resultado BacPP-28	Resultado BacPP-E28
Falsos	Exemplo 7	Não Promotor	**Promotor**	**Promotor**
Falsos	Exemplo 8	Não Promotor	Não Promotor	Não Promotor
Falsos	Exemplo 9	Não Promotor	Não Promotor	Não Promotor
Falsos	Exemplo 10	Não Promotor	Não Promotor	Não Promotor

Tabela concluída

A Tabela 21 traz os resultados obtidos na comparação entre os algoritmos para as 10 sequências reguladas pelo fator σ^{28}. Os resultados indicam que a validação de segunda etapa realizada pelo BacPP-E28 recuperou três sequências positivas (Exemplo 2, Exemplo 3 e Exemplo 6), inicialmente classificadas pelo BacPP como "Não Promotora". Além disso, ambas as ferramentas classificaram outras duas sequências como "Promotoras".

Ao avaliar os resultados gerais para essas 20 amostras, observa-se que o validador de segunda etapa apresentou um desempenho semelhante ao BacPP na classificação de sequências falsas e desempenho superior na classificação de sequências reguladas pelo fator σ^{28}. Em outras palavras, sua principal contribuição é a melhoria do desempenho na redução de falsos negativos.

Tabela 21 – Comparação entre o BacPP e BacPP-E28 - Fator σ^{28}

Fator	Sequência	Resultado Esperado	Resultado BacPP-28	Resultado BacPP-E28
σ^{28}	Exemplo 1	Promotor	Promotor	Promotor
σ^{28}	Exemplo 2	Promotor	**Não Promotor**	Promotor
σ^{28}	Exemplo 3	Promotor	**Não Promotor**	Promotor
σ^{28}	Exemplo 4	Promotor	**Não Promotor**	**Não Promotor**
σ^{28}	Exemplo 5	Promotor	**Não Promotor**	**Não Promotor**
σ^{28}	Exemplo 6	Promotor	**Não Promotor**	Promotor
σ^{28}	Exemplo 7	Promotor	Promotor	Promotor
σ^{28}	Exemplo 8	Promotor	**Não Promotor**	**Não Promotor**
σ^{28}	Exemplo 9	Promotor	**Não Promotor**	**Não Promotor**
σ^{28}	Exemplo 10	Promotor	Promotor	**Não Promotor**

A Tabela 22 apresenta os resultados obtidos na comparação entre os algoritmos para os diversos fatores σ. Observa-se que o BacPP obteve um desempenho satisfatório na classificação, enquanto o BacPP-E28 apresentou discordâncias equivocadas.

Embora os promotores sejam regulados por diferentes fatores, eles são compostos pelas mesmas quatro bases nitrogenadas (A, T, C, G), o que lhes confere certa similaridade. Na análise realizada pelo algoritmo implementado, o foco não está na composição das letras em si, mas na transformação dessas sequências em valores de estabilidade. Em termos de estabilidade, os promotores regulados por outros fatores σ apresentam alta similaridade com aqueles regulados pelo fator σ^{28}, o que contribui para a classificação equivocada realizada pelo algoritmo BacPP-E28.

O BacPP, por outro lado, apresentou desempenho mais satisfatório nesses conjuntos, pois analisa o conteúdo completo da sequência em vez de se concentrar em uma característica específica, como é o caso do algoritmo implementado neste trabalho. Devido à forma como foi concebido, considerando exclusivamente a estabilidade da sequência para a classificação, o BACpp-E28 não é adequado para uso como primeira etapa de análise.

Tabela 22 – Comparação entre o BacPP e BacPP-E28 - Demais fatores σ

Fator	**Sequência**	**Resultado Esperado**	**Resultado BacPP-28**	**Resultado BacPP-E28**
σ^{24}	Exemplo 1	Não Promotor	Não Promotor	**Promotor**
σ^{24}	Exemplo 2	Não Promotor	Não Promotor	**Promotor**
σ^{24}	Exemplo 3	Não Promotor	**Promotor**	**Promotor**
σ^{24}	Exemplo 4	Não Promotor	Não Promotor	Não Promotor
σ^{24}	Exemplo 5	Não Promotor	Não Promotor	**Promotor**
σ^{24}	Exemplo 6	Não Promotor	Não Promotor	**Promotor**
σ^{24}	Exemplo 7	Não Promotor	Não Promotor	**Promotor**
σ^{24}	Exemplo 8	Não Promotor	Não Promotor	**Promotor**
σ^{24}	Exemplo 9	Não Promotor	Não Promotor	**Promotor**
σ^{24}	Exemplo 10	Não Promotor	Não Promotor	Não Promotor
σ^{32}	Exemplo 1	Não Promotor	Não Promotor	**Promotor**
σ^{32}	Exemplo 2	Não Promotor	Não Promotor	**Promotor**
σ^{32}	Exemplo 3	Não Promotor	Não Promotor	Não Promotor
σ^{32}	Exemplo 4	Não Promotor	**Promotor**	Não Promotor
σ^{32}	Exemplo 5	Não Promotor	Não Promotor	**Promotor**
σ^{32}	Exemplo 6	Não Promotor	Não Promotor	**Promotor**
σ^{32}	Exemplo 7	Não Promotor	Não Promotor	Não Promotor
σ^{32}	Exemplo 8	Não Promotor	Não Promotor	**Promotor**
σ^{32}	Exemplo 9	Não Promotor	Não Promotor	**Promotor**
σ^{32}	Exemplo 10	Não Promotor	Não Promotor	**Promotor**

Continua na próxima página

Fator	Sequência	Resultado Esperado	Resultado BacPP-28	Resultado BacPP-E28
σ^{38}	Exemplo 1	Não Promotor	Não Promotor	Não Promotor
σ^{38}	Exemplo 2	Não Promotor	Não Promotor	Não Promotor
σ^{38}	Exemplo 3	Não Promotor	Não Promotor	Não Promotor
σ^{38}	Exemplo 4	Não Promotor	Não Promotor	Não Promotor
σ^{38}	Exemplo 5	Não Promotor	Não Promotor	Não Promotor
σ^{38}	Exemplo 6	Não Promotor	Não Promotor	**Promotor**
σ^{38}	Exemplo 7	Não Promotor	Não Promotor	**Promotor**
σ^{38}	Exemplo 8	Não Promotor	Não Promotor	Não Promotor
σ^{38}	Exemplo 9	Não Promotor	Não Promotor	Não Promotor
σ^{38}	Exemplo 10	Não Promotor	Não Promotor	**Promotor**
σ^{54}	Exemplo 1	Não Promotor	Não Promotor	**Promotor**
σ^{54}	Exemplo 2	Não Promotor	Não Promotor	**Promotor**
σ^{54}	Exemplo 3	Não Promotor	Não Promotor	Não Promotor
σ^{54}	Exemplo 4	Não Promotor	Não Promotor	**Promotor**
σ^{54}	Exemplo 5	Não Promotor	Não Promotor	Não Promotor
σ^{54}	Exemplo 6	Não Promotor	Não Promotor	Não Promotor
σ^{54}	Exemplo 7	Não Promotor	Não Promotor	Não Promotor
σ^{54}	Exemplo 8	Não Promotor	Não Promotor	Não Promotor
σ^{54}	Exemplo 9	Não Promotor	Não Promotor	**Promotor**
σ^{54}	Exemplo 10	Não Promotor	Não Promotor	**Promotor**
σ^{70}	Exemplo 1	Não Promotor	Não Promotor	**Promotor**
σ^{70}	Exemplo 2	Não Promotor	Não Promotor	**Promotor**
σ^{70}	Exemplo 3	Não Promotor	Não Promotor	**Promotor**
σ^{70}	Exemplo 4	Não Promotor	Não Promotor	Não Promotor
σ^{70}	Exemplo 5	Não Promotor	Não Promotor	Não Promotor
σ^{70}	Exemplo 6	Não Promotor	Não Promotor	Não Promotor
σ^{70}	Exemplo 7	Não Promotor	Não Promotor	**Promotor**
σ^{70}	Exemplo 8	Não Promotor	Não Promotor	**Promotor**
σ^{70}	Exemplo 9	Não Promotor	Não Promotor	**Promotor**
σ^{70}	Exemplo 10	Não Promotor	Não Promotor	Não Promotor

Tabela concluída

Esse comportamento do BacPP-E28 é esperado, uma vez que, conforme discutido anteriormente, ele foi desenvolvido com base nas características de sequências reguladas especificamente pelo fator σ^{28}. Dessa forma, a redução no desempenho em relação a outros reguladores reflete a especialização do algoritmo. Portanto, o uso do algoritmo desenvolvido como validador

de segunda etapa deverá ser restrito exclusivamente a requisições referentes ao fator σ^{28}.

No estudo de Casa *et al.* (2022), destaca-se a importância de utilizar propriedades estruturais das sequências, como curvatura e estabilidade, para uma classificação mais precisa de promotores. Embora promotores compartilhem algumas características estruturais, a tarefa de classificá-los permanece desafiadora. A análise sugere que, embora a curvatura seja uma propriedade valiosa para diferenciar promotores de outras regiões do DNA, ela, por si só, não é suficiente para distinguir com eficácia os diferentes fatores sigma. Isso ocorre porque composições distintas de nucleotídeos podem gerar estruturas semelhantes, limitando a capacidade de uma única propriedade estrutural de realizar a distinção desejada.

Por sua vez, Martinez *et al.* (2021) exploram características físico-químicas do DNA, como a estabilidade livre, e demonstram que a transformação das sequências em atributos estruturais facilita a identificação de características específicas de promotores. O estudo destaca que as sequências promotoras exibem padrões característicos de estabilidade e empilhamento, os quais são úteis para distinguir promotores de outras sequências genômicas. No entanto, os autores apontam que o uso de características isoladas é insuficiente para garantir uma classificação precisa, especialmente devido às semelhanças entre promotores regulados por diferentes fatores sigma.

Diante dessas observações, a implementação de um validador de segunda etapa que integre propriedades estruturais adicionais, como curvatura e estabilidade livre, surge como uma estratégia promissora para superar as limitações de algoritmos convencionais. Tais algoritmos, ao se basearem apenas em motivos consensuais, tendem a apresentar altos índices de falsos positivos e negativos, especialmente ao processar sequências com estruturas ou composições semelhantes às de promotores, mas que não possuem função promotora. Ao introduzir um validador que considere múltiplas propriedades estruturais, é possível aprimorar as previsões e reduzir a taxa de erros, resultando em maior precisão e confiabilidade na identificação de promotores regulados por fatores específicos, como o σ^{28}.

Em suma, a incorporação de características estruturais em uma segunda etapa de validação possibilita uma análise mais aprofundada das propriedades físico-químicas do DNA, aumentando a capacidade de discriminação entre promotores e não-promotores. Essa abordagem não só contribui para uma classificação mais precisa, mas também permite uma adaptação mais eficaz do algoritmo para aplicações específicas, como a identificação de promotores regulados por diferentes fatores sigma.

7 CONSIDERAÇÕES FINAIS

Nas seções anteriores, foram apresentados o desenvolvimento e os resultados obtidos com a implementação e validação de um algoritmo de segunda etapa para a já consolidada ferramenta de predição de promotores BacPP. Durante o processo, foram utilizadas métricas de classificação amplamente reconhecidas, como precisão, especificidade e acurácia, que permitiram uma avaliação detalhada tanto do desempenho do algoritmo na classificação correta de promotores quanto na identificação de não-promotores. A escolha dessas métricas foi essencial para garantir a confiabilidade dos resultados e fornecer uma base sólida para interpretar os desempenhos relativos do BacPP e do BacPP-E28.

A implementação seguiu o algoritmo proposto por Dall'alba (2018), utilizando a linguagem Python e a biblioteca Biopython, comumente empregada em bioinformática para a manipulação de sequências. A metodologia aplicada mostrou-se eficaz para o contexto, permitindo a modelagem precisa e a classificação eficiente de sequências reguladas pelo fator σ^{28}. A escolha de estruturas de dados, como listas aninhadas e dicionários, possibilitou o armazenamento e a manipulação eficiente dos dados, enquanto a modularidade da ferramenta garantiu que futuras modificações possam ser feitas sem impacto negativo nos processos já desenvolvidos. Essa estrutura facilitou tanto a implementação quanto a análise dos resultados, tornando a ferramenta escalável e adaptável.

Com a aplicação das métricas de validação, foi possível confirmar que o BacPP-E28 apresenta um desempenho eficiente na classificação de sequências reguladas pelo fator σ^{28}, demonstrando um equilíbrio adequado entre precisão e especificidade. Esses resultados reforçam a aplicabilidade do BacPP-E28 como um validador de segunda etapa confiável para a identificação de promotores regulados, contribuindo para o avanço da predição de promotores bacterianos e ampliando o potencial de uso da ferramenta em pesquisas bioinformáticas.

Durante o desenvolvimento deste trabalho, foram identificadas algumas oportunidades de expansão que, por limitações de tempo, não puderam ser exploradas. Uma delas é o desenvolvimento de algoritmos de validação de segunda etapa para outros fatores σ, além do σ^{28}. A inclusão de múltiplos fatores ampliaria significativamente a aplicabilidade da ferramenta, permitindo análises comparativas entre diferentes reguladores e aumentando a versatilidade da ferramenta em diversos contextos biológicos. Outro aspecto que ficou fora do escopo desta implementação foi a integração da solução desenvolvida em uma interface web, para uso direto com o BacPP.

REFERÊNCIAS

ALVES, W. P. **Linguagem e Lógica de Programação**. São Paulo, 2013. Disponível em: <https://integrada.minhabiblioteca.com.br/reader/books/9788536519371/pageid/11>. Acesso em: 03 set. 2023.

BARUAH, C.; DEKA, B.; MAHANTA, S. A review of recent advances in translational bioinformatics and systems biomedicine. In: ____. **Information Retrieval in Bioinformatics: A Practical Approach**. Singapore: Springer Nature Singapore, 2022. p. 37–62. ISBN 978-981-19-6506-7. Disponível em: <https://doi.org/10.1007/978-981-19-6506-7_3>. Acesso em: 19 ago. 2024.

BIANCO, B.; LIPAY, M. V. N. **Biologia Molecular: métodos e interpretação**. Rio de Janeiro, 2015. Disponível em: <https://integrada.minhabiblioteca.com.br/reader/books/978-85-277-2768-/epubcfi/6/10[\%3Bvnd.vst.idref\%3Dcopyright]!/4/18/10/1:20[767\%2C-9]>. Acesso em: 19 nov. 2023.

CASA, P. L. *et al.* Beyond consensual motifs: an analysis of dna curvature within escherichia coli promoters. **Biologia**, v. 77, n. 1, p. 1095–1102, 2022. Disponível em: <https://link.springer.com/article/10.1007/s11756-021-00999-0>. Acesso em: 29 out. 2023.

CASTRO, L. N. de; FERRARI, D. G. **Introdução à mineração de dados**: Conceitos básicos, algoritmos e aplicações. São Paulo, 2016. Disponível em: <https://integrada.minhabiblioteca.com.br/reader/books/978-85-472-0100-5/pageid/2>. Acesso em: 16 set. 2023.

CONCI, G. F. **Reengenharia do software BacPP.** 54 p. Dissertação (Bacharel em Ciência da Computação) — Universidade de Caxias do Sul, Bento Gonalves, 2023.

DALL'ALBA, G. **Development of an algorithm for promoter prediction of Escherichia coli using information of the DNA duplex stability for 28 -related sequences.** 18 p. Dissertação (Bacharel em Ciências Biológicas) — Universidade de Caxias do Sul, Caxias do Sul, 2018.

FACELI, K. *et al.* **Inteligencia Artificial**: Uma abordagem de aprendizado de máquina. Rio de Janeiro, 2021. Disponível em: <https://integrada.minhabiblioteca.com.br/reader/books/9788521637509/epubcfi/6/10[%3Bvnd.vst.idref%3Dcopyright]!/4>. Acesso em: 15 set. 2023.

FORBELLONE, A. L. V.; EBERSPACHER, H. F. **Logica de Programação: a construção de algoritmos e estrutura de dados**. São Paulo, 2006. Disponível em: <https://fateczlads.files.wordpress.com/2014/01/livro-lc3b3gica-de-programac3a7c3a3o-andrc3a9-luiz-villar-forbellone-e-henri-frederico-eberspc3a4cher.pdf>. Acesso em: 03 set. 2023.

GAMA-CASTRO, S. *et al.* Regulondb (version 6.0): gene regulation model of escherichia coli k-12 beyond transcription, active (experimental) annotated promoters and textpresso navigation. **Nucleic Acids Research**, v. 36, p. D120–D124, 2008. ISSN 0305-1048. Disponível em: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2238961/>. Acesso em: 29 out. 2023.

ITO, E. A. *et al.* BASiNET—BiologicAl Sequences NETwork: a case study on coding and non-coding RNAs identification. **Nucleic Acids Research**, v. 46, n. 16, p. e96–e96, 2018.

ISSN 0305-1048. Disponível em: <https://academic.oup.com/nar/article/46/16/e96/5033156>. Acesso em: 22 out. 2023.

LI, R.; MA, Z.; TIAN, S. Multiomics data analytics in agricultural breeding and cancer treatment. **Nature**, Nature Publishing Group, 2022. Disponível em: <https://www.nature.com/articles/d42473-022-00041-0>. Acesso em: 19 ago. 2024.

MARTINEZ, G. S. Ferramenta bacpp: predizendo promotores de bactérias gram-negativas e seus fatores sigma associados. In: CASA, PEDRO LENZ AND SILVA, SCHEILA DE AVILA E. **Da biologia a tecnologia**. Caxias do Sul: Não Publicado, 2023. p. 70–77.

MARTINEZ, G. S. *et al.* Beyond consensual motifs: an analysis of dna curvature within escherichia coli promoters. **SN Applied Sciences**, v. 3, 2021. ISSN 2523-3971. Disponível em: <https://link.springer.com/article/10.1007/s42452-021-04713-2#citeas>. Acesso em: 29 out. 2023.

NETTO, A.; MACIEL, F. **Python para Data Science e Machine Learning Descomplicado**. Rio de Janeiro, 2021. Disponível em: <https://integrada.minhabiblioteca.com.br/reader/books/9786555203172/epubcfi/6/4[\%3Bvnd.vst.idref\%3Dabertura.xhtml]!/4[EPUB_PythonDataScience]/16/2[_idContainer006]/2\endgroup\%4051:89>. Acesso em: 15 nov. 2023.

PRESSMAN, R. S.; MAXIM, B. R. **Metodologias Ágeis**: Engenharia de software sob medida. São Paulo, 2012. Disponível em: <https://integrada.minhabiblioteca.com.br/reader/books/9788536519418/pageid/4>. Acesso em: 15 nov. 2023.

____. **Engenharia de Software**: Uma abordagem profissional. Porto Alegre, 2021. Disponível em: <https://integrada.minhabiblioteca.com.br/reader/books/9786558040118/epubcfi/6/6[\%3Bvnd.vst.idref\%3DFR.xhtml]!/4[PRESSMAN_Completo]/2[page_iii]/2/2>. Acesso em: 15 nov. 2023.

QIN, Y. *et al.* From the perspective of computer engineering:: Research on digestive diseases based on bioinformatics and genetic genetics: – research and development of an innovative economic product of computer medicine. **Highlights in Science, Engineering and Technology**, v. 39, p. 1441–1446, Apr. 2023. Disponível em: <https://drpress.org/ojs/index.php/HSET/article/view/6932>. Acesso em: 18 nov. 2023.

QUEROZ, R. S. de *et al.* As aplicabilidades da bioinformática na área da saúde. In: FóRUM RONDONIENSE DE PESQUISA, VIII., 2022, Rondônia. **Afya**. Rondônia: Afya, 2022. Forum. Disponível em: <https://periodicos.saolucasjiparana.edu.br/foruns/article/view/540/506>. Acesso em: 15 nov. 2023.

ROSA, A. C. F. d. *et al.* Uso de técnicas de aprendizado de máquina para classificação de fatores que influenciam a ocorrência de dermatites ocupacionais. **Revista Brasileira de Saúde Ocupacional**, Fundação Jorge Duprat Figueiredo de Segurança e Medicina do Trabalho - FUNDACENTRO, v. 48, p. e4, 2023. ISSN 0303-7657. Disponível em: <https://doi.org/10.1590/2317-6369/31620pt2023v48e4>. Acesso em: 01 nov. 2023.

RUSSEL, S.; NORVIG, P. **Inteligencia Artificial**: Uma abordagem moderna. Rio de Janeiro, 2022. Disponível em: <https://integrada.minhabiblioteca.com.br/reader/books/9788595159495/epubcfi/6/60[%3Bvnd.vst.idref%3Dchapter19]!/4/764/1:27[zer%2Co:]>. Acesso em: 07 set. 2023.

SILVA, J. C. F. da *et al.* Using mobile edge ai to detect and map diseases in citrus orchards. **Sensors**, v. 23, n. 4, 2023. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/23/4/2165>. Acesso em: 09 out. 2023.

SILVA, L. A. da; PERES, S. M.; BOSCARIOLI, C. **Introdução à Mineração de Dados**: Com aplicações em r. Rio de Janeiro, 2016. Disponível em: <https://integrada.minhabiblioteca.com.br/reader/books/9788595155473/epubcfi/6/6[%3Bvnd.vst.idref%3Dcreditos.html]!/4/68/2[table-1]/2/4/4>. Acesso em: 16 set. 2023.

SILVA, S. d. A.; ECHEVERRIGARAY, S.; GERHARDT, G. J. Bacpp: Bacterial promoter prediction—a tool for accurate sigma-factor specific assignment in enterobacteria. **Journal of Theoretical Biology**, v. 287, p. 92–99, 2011. ISSN 0022-5193. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0022519311003675>. Acesso em: 29 out. 2023.

SILVA, S. d. A. e. **Redes neurais artificiais aplicadas no reconhecimento de regiões promotoras em bactérias Gram-negativas**. Tese (Doutorado) — Universidade de Caxias do Sul, Caxias do Sul, RS, Brasil, 2011. Tese de Doutorado, Programa de Pós-Graduação em Biotecnologia.

SILVA, S. d. A. e. *et al.* Dna duplex stability as discriminative characteristic for escherichia coli 54- and 28- dependent promoter sequences. **Biologicals**, v. 42, n. 1, p. 22–28, 2014. ISSN 1045-1056. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1045105613001152>. Acesso em: 29 out. 2023.

SILVA, S. d. A. e.; GERHARDT, G. J.; ECHEVERRIGARAY, S. Rules extraction from neural networks applied to the prediction and recognition of prokaryotic promoters. **Genetics and Molecular Biology**, Sociedade Brasileira de Genética, v. 34, n. 2, p. 353–360, 2011. ISSN 1415-4757. Disponível em: <https://www.scielo.br/j/gmb/a/VLDShYWJ3kRSb7Kjs8F37gr/?lang=en#>. Acesso em: 02 out. 2023.

SOUZA, M. A. F. de *et al.* **Algoritmos e Lógica de Programação**. São Paulo, 2020. Disponível em: <https://integrada.minhabiblioteca.com.br/reader/books/9788522128150/pageid/3>. Acesso em: 03 set. 2023.

SÁ, C. da C. **Métodos de validação tradicional e temporal aplicados à avaliação de classificadores de RNAS codificantes e não codificantes.** São Paulo: Universidade de São Paulo, 2018. Disponível em: <https://teses.usp.br/teses/disponiveis/95/95131/tde-19052018-122805/publico/CostaSaClebiano2018.pdf.> Acesso em: 18 nov. 2023.

TRIPATHI, L.; ZHANG, Y.; LIN, Z. Bacterial sigma factors as targets for engineered or synthetic transcriptional control. **Frontiers in Bioengineering and Biotechnology**, v. 2, 2014. ISSN 2296-4185. Disponível em: <https://www.frontiersin.org/journals/bioengineering-and-biotechnology/articles/10.3389/fbioe.2014.00033>. Acesso em: 19 ago. 2024.

ANEXO A – IMPLEMENTAÇÃO DE CÓDIGO

1 – Implementação completa do BACpp -E28

```
# biblioteca especifica para leitura de arquivo fasta
from Bio import SeqIO
import csv

# letras validas para sequencia
letrasValidas = ['A', 'T', 'C', 'G']

#dicionario com a conversao dos pares de nucleotideos para valor numerico
regraConversao = {
    'AA': -1.00, 'AT': -0.88, 'TA': -0.58, 'AG': -1.30, 'GA': -1.30,
    'TT': -1.00, 'AC': -1.45, 'CA': -1.45, 'TG': -1.44, 'GT': -1.44,
    'TC': -1.28, 'CT': -1.28, 'CC': -1.84, 'CG': -2.17, 'GC': -2.24,
    'GG': -1.84
}

# funcao que valida os valores da sequencia
def validaSequencia(seq):
    for caractere in seq:
        # valida se a sequencia possui pelo menos 80 letras
        if len(seq) < 81:
            print(
                "Sequ ncia   menor que o permitido! " +
                "Tamanho m nimo permitido: 81 | Tamanho atual:", len(seq))
            return 0

        # valida se a sequencia possui somente as letras validas
        if caractere.upper() not in letrasValidas:
            print("Sequ ncia inserida possui valor inv lido!")
            return 0
        else:
            return 1

# funcao que trata a entrada da sequencia
def inputSequencia(arquivo):
    listaSequencias = []
    listaCabecalho = []

    aux = 0

    while aux == 0:
```

```
        aux = 1
        for registro in SeqIO.parse(arquivo, "fasta"):
            cabecalho = registro.id
            sequencia = str(registro.seq).upper()
            if (validaSequencia(sequencia)) == 1:
                listaSequencias.append(sequencia)
                listaCabecalho.append (cabecalho)
            else:
                aux = 0  # retorna para o menu


    return listaCabecalho, listaSequencias

# funcao que faz a divisao das sequencias
def trataEntrada(sequencia):
    sequenciaTratada = []

    for seqLidas in sequencia:
        # variavel para armazenar a diferen a entre o
        # tamanho da sequencia e as 80 posicoes
        diferenca = 0

        # variavel para armazenar o tamanho da lista
        tam = len(seqLidas)
        # lista para armazenar a sequencia apos a divisao
        sequenciasDivididas = []

        # Se a sequencia tiver exatamente as 81 posicoes
        # nao e necessario fazer o laco
        if (len(seqLidas) >= 81):
            diferenca = len(seqLidas) - 81

        if (diferenca > 0):
            for i in range(diferenca+1):
                sequenciaNova = seqLidas[i:i+80]
                sequenciasDivididas.append(sequenciaNova)

        else:
            # se a sequencia possuir 80 posicoes armazena direto.
            sequenciasDivididas.append(seqLidas)


        sequenciaTratada.append(sequenciasDivididas)

    return sequenciaTratada

# funcao que converte o par de nucleotideo em valor
```

```
def converteNucleotideo(listaSequencias):
    # lista que ira armazenar os itens de cada
    # sublistas da lista recebida como parametro
    listaConvertida = []

    for sequencia_i in listaSequencias:
        # lista que ira armazenar todas as strings de cada sublista
        convSublista = []

        for string in sequencia_i:
            tamanho = len(string)
            # lista que ira armazenar a string (lista menor)
            # convertida em valores.
            sequenciaConvertida = []

            for pos in range(tamanho - 1):
                par = string[pos:pos + 2]
                if par in regraConversao:
                    resultadoAux = regraConversao[par]

                sequenciaConvertida.append(resultadoAux)

            convSublista.append(sequenciaConvertida)

        listaConvertida.append(convSublista)

    return listaConvertida

# funcao que identifica os componentes principais
def componentePrincipal(item, pos):
    # PC1
    if pos >= 46 and pos <= 50:
        resultado = item - (-1.67)
    # PC2
    elif ((pos >= 11 and pos <= 16) or (pos >= 18 and pos <= 21) or
          (pos >= 27 and pos <= 31) or (pos >= 51 and pos <= 54) or
          (pos >= 71 and pos <= 74)):
        resultado = item - \
            (((-1.42) + (-1.24) + (-1.16) + (-1.05) + (-1.36)) / 5)
    # PC3
    elif (pos >= 35 and pos <= 39):
        resultado = item - (-1.42)
    else:
        resultado = 0

    return resultado
```

```
#funcao que calcula a diferenca entre a regra e a sequencia
def calculaDiferenca(sequenciaConvertida):
    listaDiferenca = []
    for sequenciaConvertida_i in sequenciaConvertida:
        diferencaInterna = []
        for item in sequenciaConvertida_i:
            listaDiferencaCalculada = []
            for componente, indice in enumerate(item, start = 1):
                listaDiferencaCalculada.append(
                    componentePrincipal(indice, componente))
            diferencaInterna.append(listaDiferencaCalculada)
        listaDiferenca.append(diferencaInterna)

    return listaDiferenca

#funcao que soma o valor de cada posicao da sequencia
def soma(sequencia):
    sequenciaResultado = []
    for sequencia_i in sequencia:
        resultadoAgrupado = []
        for valor in sequencia_i:
            resultado = 0
            for i in valor:
                resultado = resultado + i
            resultadoAgrupado.append(round (resultado,2))
        sequenciaResultado.append(resultadoAgrupado)

    return sequenciaResultado

#funcao que classifica o gene de forma literal
def classificaGene(resultadoSoma):
    finalLiteral = []
    for agrupamentoResultado in resultadoSoma:
        geneAgrupado = []
        for resultadoSoma_i in agrupamentoResultado:
            if resultadoSoma_i >= 0:
                gene = 'NAO_PROMOTOR'
            else:
                gene = 'PROMOTOR'
            geneAgrupado.append(gene)
        finalLiteral.append(geneAgrupado)

    return finalLiteral

#funcao que escreve os arquivos de logs.
def escreveArquivos(cabecalho, sequencia, sequenciaTratada,
                    sequenciaConvertida, diferenca,
```

```
                    final,  finalLiteral):

# Garantindo que todas as listas tenham o mesmo tamanho
  tamanhoMinimo = min(len(sequenciaTratada),
                      len(sequenciaConvertida),
                      len(diferenca), len(final),
                      len(finalLiteral))

  # gerando arquivo com as sequencias divididas
  with open("Logs/gerado.txt", "w") as arquivoGerado:
      for sequenciaNova in sequenciaTratada:
          arquivoGerado.write("\n")
          for seq in sequenciaNova:
              arquivoGerado.write(str(seq) + "\n")

  # gerando arquivo com os valores convertidos
  with open("Logs/sequenciaConv.txt", "w") as arqSequenciaConv:
      for seq in sequenciaConvertida:
          arqSequenciaConv.write("\n")
          arqSequenciaConv.write("******************************" +
                                 "******************************␣\n")
          for item in seq:
              arqSequenciaConv.write(
                  "Sequencia␣Convertida:\n␣" + str(item) + "\n\n")

  # gerando arquivo com as diferen as calculadas
  with open("Logs/diferenca.txt", "w") as arqDif:
      for diferenca_i in diferenca:
          arqDif.write("\n")
          arqDif.write("**********************************␣" +
                       "**************************␣\n")
          for d in diferenca_i:
              arqDif.write(
                  "Diferenca␣calculada␣:\n␣" + str(d) + "\n\n")

   # gerando arquivo com o resultado da soma
  with open("Logs/final.txt", "w") as arqFinal:
      for final_i in final:
          arqFinal.write("\n")
          arqFinal.write("**********************************␣" +
                         "**************************␣\n")
          for r in final_i:
              arqFinal.write(
                  "Resultado␣final␣:\n␣" + str(r) + "\n\n")

  # gerando arquivo com a classificacao
  with open("Logs/finalLiteral.txt", "w") as arqFinalLit:
```

```
        for finaliteral_i in finalLiteral:
            arqFinalLit.write("\n")
            arqFinalLit.write("**********************************␣" +
                              "**************************␣\n")
            for fl in finaliteral_i:
                arqFinalLit.write(
                    "Resultado␣final␣:\n␣" + str(fl) + "\n\n")

    # gerando arquivo com o log final
    with open("Logs/resultadoLog.txt", "w") as log:
        # Percorre os indices das listas principais
        for x in range(tamanhoMinimo):
            log.write("\n")
            log.write("**********************************␣" +
                      "**************************␣\n")
            # Percorre os indices da sublista
            for i in range(len(sequenciaTratada[x])):
                linha = (
                    f"Tratada:␣{sequenciaTratada[x][i]}\n"
                    f"Convertida:␣{sequenciaConvertida[x][i]}\n"
                    f"Diferen a:␣{diferenca[x][i]}\n"
                    f"Soma:␣{final[x][i]}\n"
                    f"Resultado:␣{finalLiteral[x][i]}\n\n"
                )
                log.write(linha)


    with open("Logs/resultadoFinal.txt", "w") as resultado_final:
        # Percorre os indices das listas
        for x in range(tamanhoMinimo):
            resultado_final.write("**********************************␣" +
                      "**************************␣\n")

            # Exibir a sublista correspondente apos cada
            # "*******************************"
            lida = (f"Sequ ncia␣lida:␣{sequencia[x]}\n\n")
            resultado_final.write(lida)

            # Itera pelos elementos da sublista atual
            for i in range(len(sequenciaTratada[x])):
                result = (
                    f"Parte␣avaliada:␣{sequenciaTratada[x][i]}\n"
                    f"Gene:␣␣{finalLiteral[x][i]}\n\n"
                )
                resultado_final.write(result)

    # Geracao das sequencias e gravacao em CSV
```

```
    with open('Logs/resultadosObtidos.csv', 'w', newline='') as
        resultadoCsv:
        writer = csv.writer(resultadoCsv)

        # Escreve o cabe alho do CSV
        writer.writerow(["Fator_Sigma", "Sequencia", "Resultado", "Gene"])

        for x in range(tamanhoMinimo):
            for i in range(len(sequenciaTratada[x])):
                fatorSigma = cabecalho[x]
                sequenciaAvaliada = sequenciaTratada[x][i]
                resultado = final[x][i]
                resultadoLiteral = finalLiteral[x][i]
                # Salva  as informacoes no CSV
                writer.writerow([fatorSigma, sequenciaAvaliada, resultado,
                    resultadoLiteral])


# funcao principal
def main():

    # definicao das listas:
    cabecalho = []
    sequencia = []
    sequenciaTratada = []
    sequenciaConvertida = []
    diferenca = []
    final = []
    finalLiteral = []

    # entrada da sequencia:
    arquivoLido = "Sequencias/Validacao2/todos.fasta"
    #arquivoLido = "SequenciasExemplo/teste.fasta"



    cabecalho, sequencia = inputSequencia(arquivoLido)

    # tratando a entrada recebida:
    sequenciaTratada = trataEntrada(sequencia)

    # convertendo sequencia obtida apos tratamento nos valores de regras
    sequenciaConvertida = converteNucleotideo(sequenciaTratada)

    # calcula diferenca entre a regra e sequencia avaliada
    diferenca = calculaDiferenca(sequenciaConvertida)
```

```
    # calcula a soma de cada valor da sequencia.
    final = soma (diferenca)

    # gravando a classificacao obtida na lista.
    finalLiteral = classificaGene(final)

    escreveArquivos(cabecalho, sequencia, sequenciaTratada,
                    sequenciaConvertida, diferenca,  final,  finalLiteral)


if __name__ == '__main__':
    main()
    print("Processo finalizado!")
```