Estudo comparativo entre algoritmos implementados de classificação e clusterização

Daniel Vieira, Dimitri Leandro, Gabriel Fernandes, Yan Podkorytoff

[1]Centro de Engenharia, Modelagem e Ciências Sociais Aplicadas da UFABC
Av. dos Estados, 5001, Santo André, SP

Resumo. *O presente projeto aborda a implementação dos algoritmos* ***K-Nearest Neighbors****,* ***Perceptron*** *e* ***K-Means****, apresentados no decorrer do curso de Sistemas Inteligentes da UFABC, a fim de compreender como estes algoritmos funcionam e como são implementados em linguagem de programação. Em seguida, propõe-se a comparação entre os algoritmos implementados neste estudo e as implementações disponíveis na biblioteca* ***Scikit-Learn*** *[Pedregosa et al. 2011] com o auxílio das bases de dados* ***Iris Plants Dataset*** *[Fisher 1936],* ***Olivetti Faces Dataset*** *[Cambridge University Engineering Department 1992],* ***Breast Cancer Wisconsin Dataset*** *[W.N. Street and Mangasarian 1993] e* ***Wine Recognition Dataset*** *[Lichman 2013].*

1. Introdução

Há quatro principais abordagens em aprendizado de máquinas: classificação, *clusterização*, regressão e otimização, dentre as quais pode-se separar dois principais tipos de aprendizado: supervisionado e não-supervisionado.

No processo de aprendizado supervisionado, um dos algoritmos mais simples que pode ser encontrado na literatura é o *K-Nearest Neighbors*, também conhecido como KNN, amplamente utilizado no processo de classificação [Altman 1992]. Dado um conjunto de entradas previamente classificadas, o algoritmo KNN aborda o conceito de proximidade para avaliar a classificação mais provável de um dado não rotulado.

Outro algoritmo utilizado em classificação, o *Perceptron* [Rosenblatt 1957], representa o elemento básico de processamento em redes neurais artificiais, ou seja, a mais simples arquitetura de uma rede neural artificial. Cada *Perceptron* é um sistema que processa múltiplas entradas e responde com uma saída binária, que deve ser projetada para ser determinada segundo algum valor limiar [Alpaydin 2009].

Um processo bastante utilizado de aprendizado não-supervisionado, o algoritmo *K-Means* [MacQueen 1967], é um método de *clusterização* que, ao contrário da classificação, recebe como entrada dados não rotulados e os agrupa em *clusters*. Cada *cluster* indica a possibilidade de que seus dados possuam alguma similaridade. Assim como o KNN, o *K-means* baseia-se nas distâncias euclidianas para identificar os grupos.

Neste estudo, pretende-se compreender a teoria de cada um dos algoritmos citados, identificar possíveis vantagens e desvantagens e avaliar os resultados a partir de algumas métricas importantes, tais como: acurácia, *F1 Score*, precisão, *recall*, *WB Index*, coeficiente de silhueta e *Variance Ratio Criterion*.

2. Implementações

Essa seção irá abordar o funcionamento dos algoritmos e explicar como se deram suas implementações.

2.1. K-Nearest Neighbors

O algoritmo não-paramétrico conhecido como K-Nearest Neighbors, ou simplesmente **KNN**, é um dos principais e mais simples algoritmos utilizados no processo de classificação em *Machine Learning*. Seu método de classificação baseia-se no conceito de distância entre um objeto não classificado e outras classes de uma base de dados. A proximidade indica sua pertinência em um conjunto já classificado. Este tipo de processo é conhecido como aprendizagem supervisionada.

O funcionamento do algoritmo KNN é esquematizado no fluxograma da figura 1.

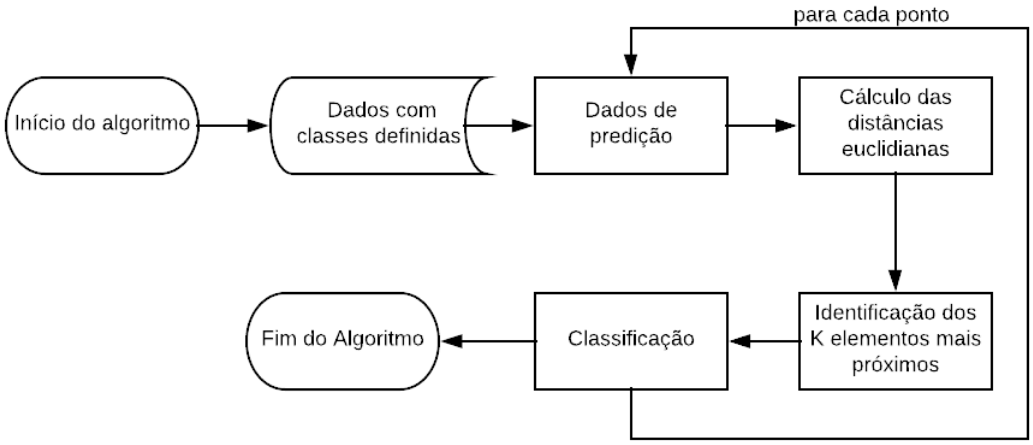

Figura 1. Fluxograma representativo do algoritmo K-Nearest Neighbors. Elaborado pelos autores.

De acordo com o fluxograma na figura 1, o dados de treinamento são armazenados com suas respectivas classes. A partir do conjunto de dados de testes, o classificador calcula as distâncias euclidianas em relação aos demais dados classificados, conforme a equação , verifica os K elementos mais próximos e os classifica de acordo com a classe mais provável. Segundo [Murphy 2012], a equação 2 define formalmente uma classificação de acordo com os pares contidos no conjunto $D = (x_i, y_i)_{i=1}^{K}$ tal que K é a quantidade de pontos mais próximos, x_i representa o dado de entrada e y_i sua categoria.

$$d = \sqrt{\sum_{i=0}^{N}(x_i - y_i)^2} \tag{1}$$

$$p(y = x|x, D, K) = \frac{1}{K}\sum I(y_i = c) \tag{2}$$

Sendo I definido pela função 3, abaixo:

$$I(e) = \begin{cases} 1, \text{se } e \text{ for verdadeiro} \\ 0, \text{se } e \text{ for falso} \end{cases} \tag{3}$$

A figura 2 ilustra o modelo KNN de classificação na qual o valor de K pode ser entendido como o raio que encerra os dados mais próximos. Um ponto importante a ser levado em consideração é que um baixo valor de K pode acarretar em um *overfitting* do modelo, dependendo da base de dados. Além disso, cada dado a ser classificado precisa ser comparado a cada uma das classificações, ou seja, datasets com muitas classes podem ser um problema.

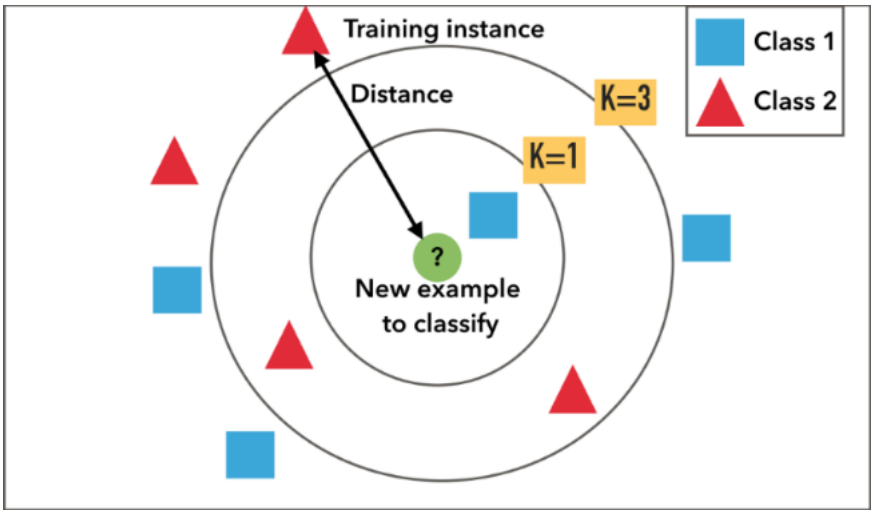

Figura 2. Representação do paradigma de classificação do algoritmo K-Nearest Neighbors. Disponível em: https://goo.gl/cZ7qmA.

2.2. Perceptron

O segundo algoritmo abordado nesse projeto também é uma ferramenta de aprendizagem supervisionada. O *Perceptron*, desenvolvido pela primeira vez por Frank Rosenblatt em 1957, consiste na separação linear de dados pertencentes a duas classes distintas. Ou seja, trata-se de um classificador binário [Rosenblatt 1957].

Durante toda sua performance, o *Perceptron* mantém um vetor de parâmetros multiplicadores w com a mesma dimensionalidade dos dados empregados, além de mais um valor de bias ao final. Seu funcionamento é apresentado a seguir:

$$y = \sum_{j=1}^{d} w_j \cdot x_j + w_0 \tag{4}$$

Na $n_{ésima}$ iteração, o algoritmo recebe uma instrução para interpretar um dado x_n e classificá-lo. A operação, então, realiza o produto vetorial entre o vetor x_n e o vetor de pesos, lembrando-se de somar o valor de bias ao final, como mostra a equação 4 [Alpaydin 2009]. Com o resultado auferido, aplica-se a equação 5 a fim de predizer

a classificação do elemento. Caso a classificação real do componente seja conhecida, pode-se usá-la a fim de atualizar o vetor de pesos e, assim, melhorar as classificações futuras [Smola and Vishwanathan 2008]. Além do vetor de pesos, é crucial determinar um parâmetro inicial geralmente mencionado como "taxa de aprendizagem" que será elucidado mais adiante.

$$f(x) = \begin{cases} 1, \text{caso } \langle w \times x \rangle + \ w_0 \geq 0 \\ 0, \text{caso contrário} \end{cases} \tag{5}$$

O *Perceptron* é constantemente referido como um algoritmo *online*. Isto é, ele é capaz de se modificar conforme o processamento de novos dados de entrada ocorre. Isso faz com que esse algoritmo possa classificar um mesmo dado de forma divergente conforme seu vetor de pesos é atualizado a cada iteração [Borodin and El-Yaniv 1998].

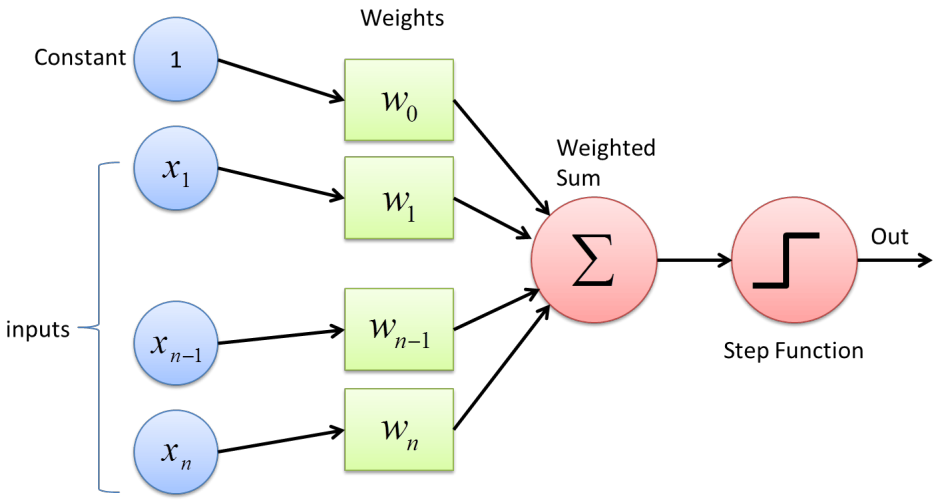

Figura 3. Representação do paradigma de classificação de um *Perceptron*. Disponível em: https://goo.gl/XGhWH7

Assim como retratado acima, o código implementado pela equipe neste projeto também dispõe de um vetor de pesos e taxa de aprendizagem. Seu principal objetivo é determinar os melhores valores possíveis para os componentes desse vetor, como se sucede:

O algoritmo é inicializado quando uma série de dados de treinamento com d dimensões é recebida. O programa, portanto, cria um vetor de pesos aleatórios entre -1 e 1 com d dimensões, incluindo o valor de bias ao final. Em seguida, confere-se a predição dos dados de treinamento com o vetor de pesos atual. Se a configuração dos pesos já produz uma acurácia de 100%, então o algoritmo é interrompido. Senão, para cada dado, deve-se calcular o erro em relação à classificação correta, definido como a subtração do valor da classificação real pelo valor da predição. Então, o valor dos pesos é recalculado de acordo com a equação 6, onde $\vec{w}$ denota o vetor de pesos, α a taxa de aprendizagem e $\vec{x}$ o dado da iteração em questão. Ao final, o algoritmo regressa à verificação da acurácia e repete o passo a passo detalhado. Ainda é possível definir um número máximo de iterações até que o programa seja interrompido mesmo que o vetor de

pesos ainda não tenha atingido sua conformação ótima [Smola and Vishwanathan 2008] [Shalev-Shwartz and Ben-David 2014].

$$\vec{w} = \vec{w} + \alpha \cdot \text{erro} \cdot \vec{x} \tag{6}$$

O parâmetro α – *Learning Factor* ou *Learning Rate* [Shalev-Shwartz and Ben-David 2014] – é uma variável multiplicadora instruída para controlar o ajuste dos valores do vetor w. Por isso, é frequentemente mencionada como taxa de aprendizagem do *Perceptron*. Em outras palavras, esse fator busca minimizar a função erro do algoritmo, almejando otimizar a função 6 em um processo conhecido como *Método do Máximo Declive* (*Stochastic Gradient Descent*) [Alpaydin 2009]. Quanto menor o valor de α, mais lentamente o vetor de pesos converge para sua configuração terminal. Mas, apesar da vagarosidade, manter esse valor no nível mais baixo possível garante que o método do máximo declive não perca nenhum ponto de mínimo na função erro [Data Science Academy 2018].

A figura 4 mostra um fluxograma básico explicando o desenvolvimento do algoritmo do perceptron.

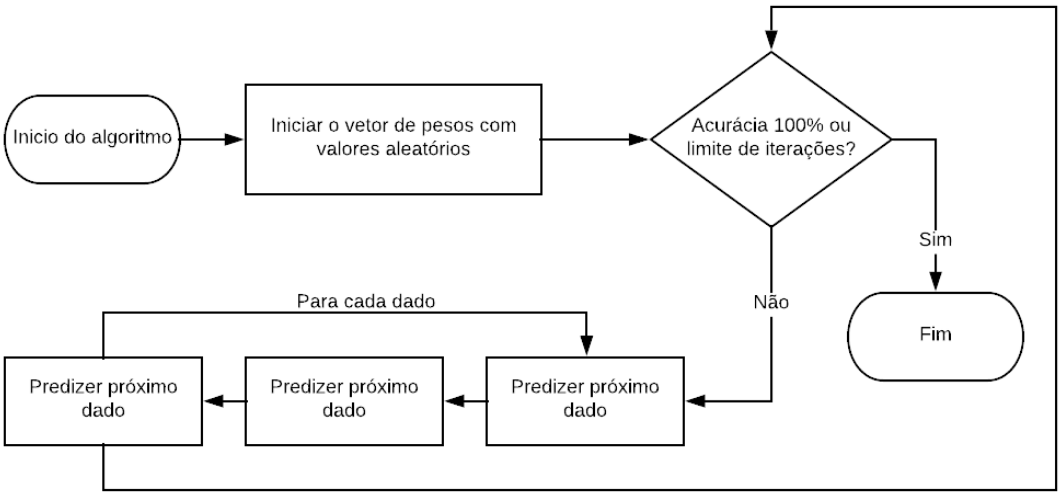

Figura 4. Fluxograma representativo do algoritmo do Perceptron. Elaborado pelos autores.

2.3. K-Means

Nos dois algoritmos abordados anteriormente tem-se um caso de aprendizagem supervisionada, o que significa dizer que a classificação correta dos dados, bem como a nomenclatura precisa das classes, é conhecida. O K-Means [MacQueen 1967] distingue-se dos algoritmos relacionados à definição acima pois é alusivo a um método de aprendizagem dito não supervisionado, intitulado *clusterização* [Alpaydin 2009].

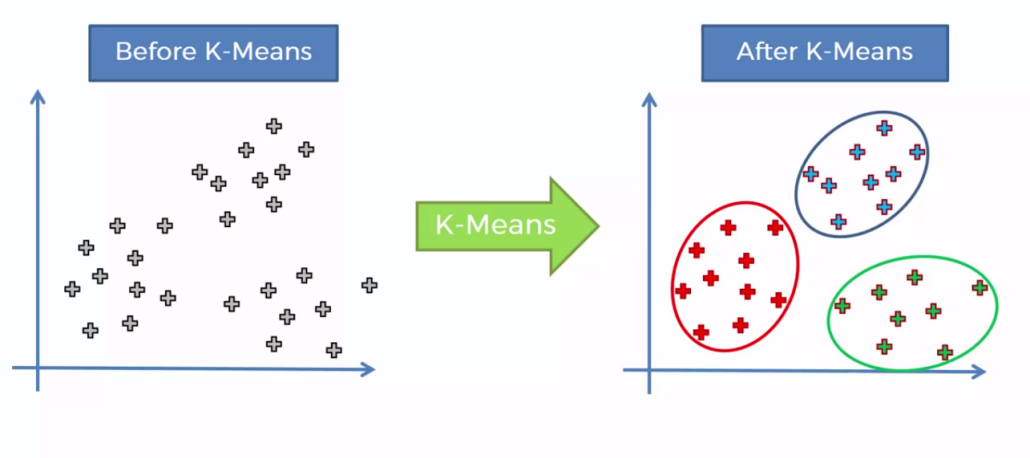

Figura 5. Representação do paradigma de *clusterização* do *K-Means*. Disponível em: https://bit.ly/2PKKt51

O K-Means visa minimizar a variância dos dados dentro de um mesmo *cluster* ao mesmo tempo que busca maximizar esse parâmetro entre os centroides de cada grupo [Alpaydin 2009] [Smola and Vishwanathan 2008]. Em outras palavras, ele pretende fazer com que dados de um mesmo *cluster* sejam mais parecidos quanto for possível entre si e mais diferentes em relação a dados de outros *clusters*.

A figura 6 exibe um fluxograma explicativo sobre o desenvolvimento desse algoritmo.

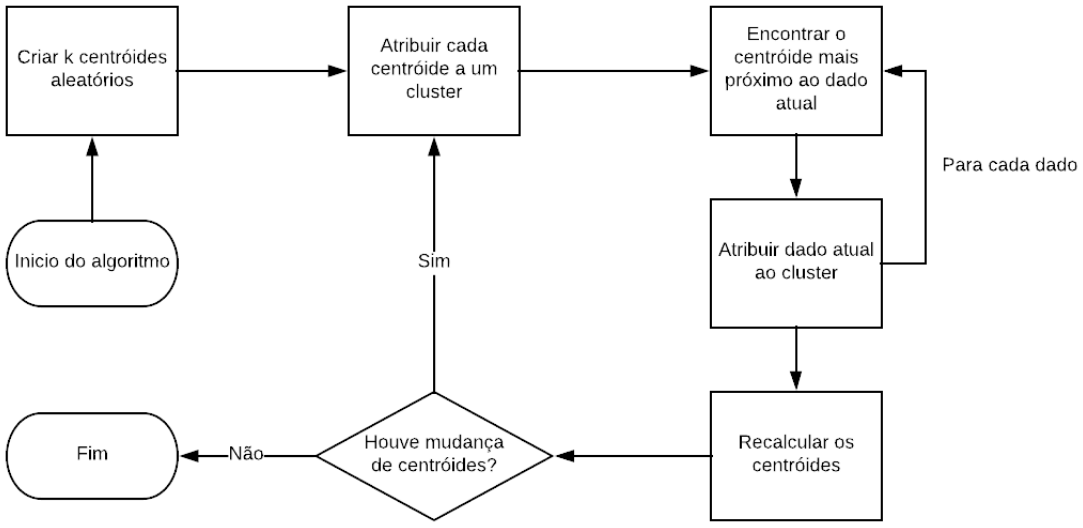

Figura 6. Fluxograma representativo do algoritmo K-Means. Elaborado pelos autores.

A princípio, o algoritmo implementado pela equipe seleciona aleatoriamente k pontos pertencentes aos dados de entrada x para serem os centroides de cada um dos k *clusters*. Em seguida, o programa atribui para cada dado x_n um *cluster*, usando a

menor distância euclidiana entre esse ponto e os centroides definidos anteriormente para concluir essa tarefa. Logo após, o centroide de cada grupo é recalculado, fazendo, para cada dimensão, a média aritmética de todos os dados pertencentes ao *cluster* em questão. As últimas duas instruções são repetidas até que em algum momento do código não haja mudança nos centroides. Portanto, encerra-se o algoritmo.

É importante ressaltar que o resultado obtido pode ser prejudicado conforme a seleção aleatória dos k centroides iniciais. Por isso, é vantajoso executar diversas iterações desse algoritmo para que só então se retorne o melhor agrupamento possível. Na prática, em cada uma dessas iterações é necessário calcular alguma métrica que examine o requinte da *clusterização* obtida a fim de atender à condição anterior. Dessa forma, o grupo optou por implementar a métrica *WB Index* [Zhao 2012], não disponível na biblioteca *Scikit Learn*.

2.4. WB Index

Como esclarecido previamente, será necessário utilizar alguma métrica para avaliar a qualidade da *clusterização* em cada iteração do K-Means com o objetivo de conhecer o melhor agrupamento produzido.

Para realizar esse encargo, o algoritmo K-Means presente na biblioteca *Scikit Learn* usa a inércia dos dados, definida em seu próprio *site* como sendo a soma das distâncias ao quadrado de cada amostra ao seu centroide [Pedregosa et al. 2011]. Entretanto, a equipe optou por implementar a métrica de *clusterização* intitulada *WB Index*, proposta pela primeira vez em 2012 em uma dissertação acadêmica da Universidade da Finlândia Oriental [Zhao 2012].

A técnica consiste em multiplicar a quantidade de dados de entrada x pela divisão entre a soma dos quadrados no interior dos clusters (*SSW: sum-of-squares within*) pela soma dos quadrados entre os clusters (*SSB: sum-of-squares between*), como mostra a equação 7.

$$WB = n_x \cdot \frac{SSW}{SSB} \tag{7}$$

Os parâmetros SSW e SSB estão definidos pelas equações 8 e 9, respectivamente. Antes, é preciso estabelecer que um determinado *cluster* i detém n_i elementos e que c_i representa seu centroide.

$$SSW = \sum_{i=1}^{k} n_i \cdot ||x_i - c_i||^2 \tag{8}$$

$$SSB = \sum_{i=1}^{k} n_i \cdot ||\bar{X} - c_i||^2 \tag{9}$$

Conforme explicitado preliminarmente, os algoritmos de *clusterização* pretendem reduzir a variância dos dados inclusos em um mesmo cluster enquanto acrescem a variância entre os próprios agrupamentos. Portanto, para um bom resultado de

clusterização, espera-se que *SSW* seja mais baixo que *SSB*, produzindo um *WB Index* tão pequeno quanto possível.

3. Resultados e Comparações

Essa seção irá detalhar os resultados obtidos e comparar os algoritmos implementados com os já disponíveis na biblioteca *Scikit Learn* através de métricas objetivas.

3.1. K-Nearest Neighbors

O primeiro algoritmo abordado será o *K-Nearest Neighbors* (KNN). Ele foi testado usando dois *datasets* distintos: *Olivetti Faces Dataset* [Cambridge University Engineering Department 1992] e *Iris Dataset* [Fisher 1936]. Além disso, para uma avaliação confiável das métricas, empregou-se o método de validação cruzada *K-Fold* com 10 pacotes. Isto é, a cada iteração, 90% dos dados serão utilizados para treinamento e o restante para teste.

3.1.1. Olivetti Faces Dataset

Esse *dataset* consiste em um conjunto de 400 amostras de dados com 4096 dimensões. Cada dado é um retrato que pode ser classificado dentre 40 pessoas distintas, isto é, 40 classes [Cambridge University Engineering Department 1992].

A figura 7, abaixo, mostra o valor da acurácia obtida em cada iteração do *K-Fold*. É possível perceber que o algoritmo da biblioteca *Scikit Learn* obteve um desempenho notoriamente superior.

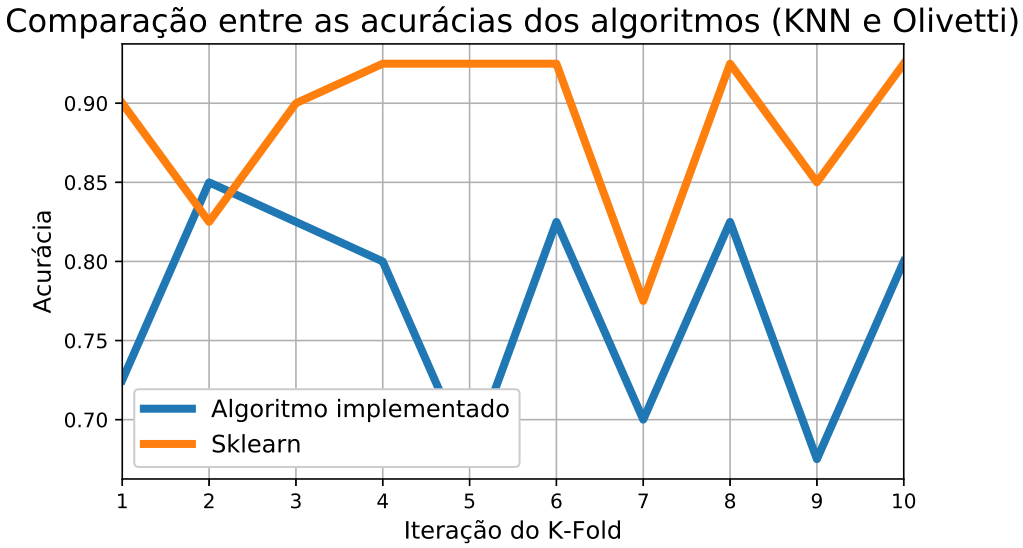

Figura 7. Acurácia obtida em cada iteração do método de validação cruzada K-Fold. Elaborado pelos autores.

A figura 8, abaixo, mostra o *F1 Score* médio entre todas as classes para cada iteração do *K-Fold*. O algoritmo da biblioteca *Scikit Learn* continua tendo um desempenho maior.

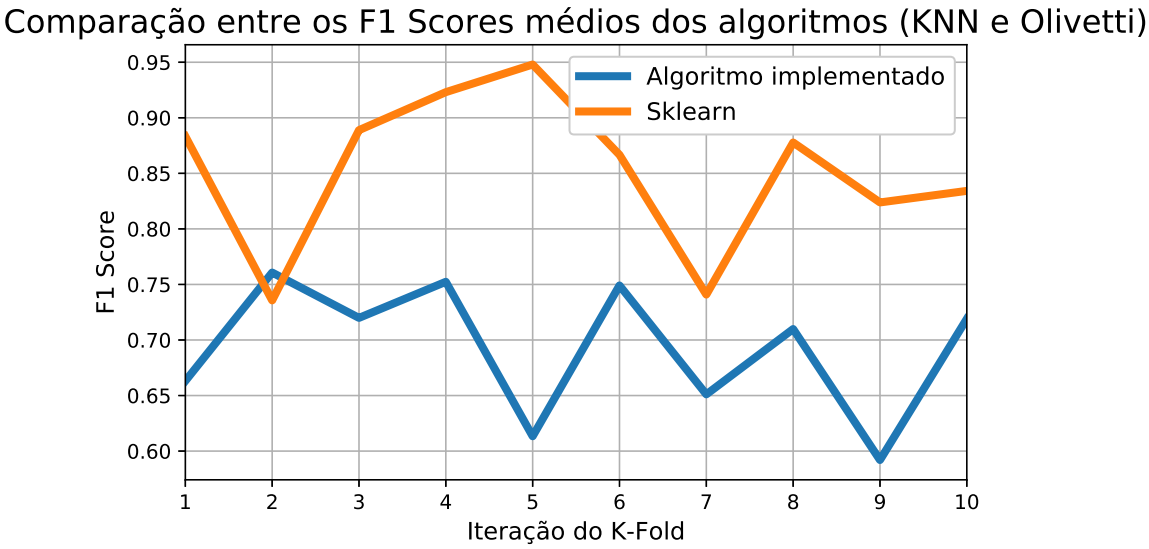

Figura 8. Média do F1 Score obtida em cada iteração do método de validação cruzada K-Fold. Elaborado pelos autores.

Analogamente, a figura 9 mostra a média da precisão das classes para cada iteração do *K-Fold*.

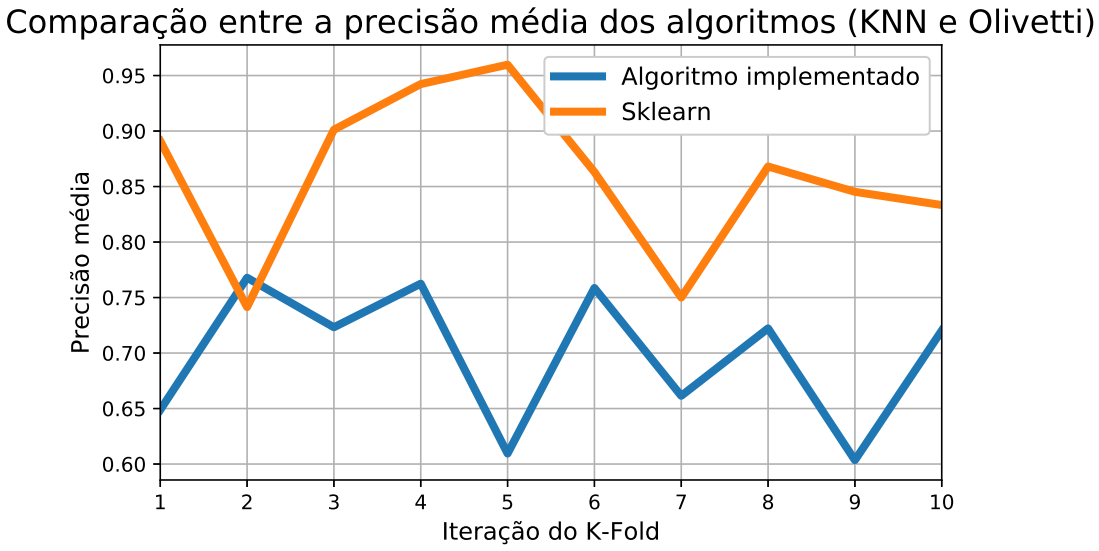

Figura 9. Precisão média obtida em cada iteração do método de validação cruzada K-Fold. Elaborado pelos autores.

A mesma diferença de performance é verificada também durante a avaliação do *Recall*, como se segue:

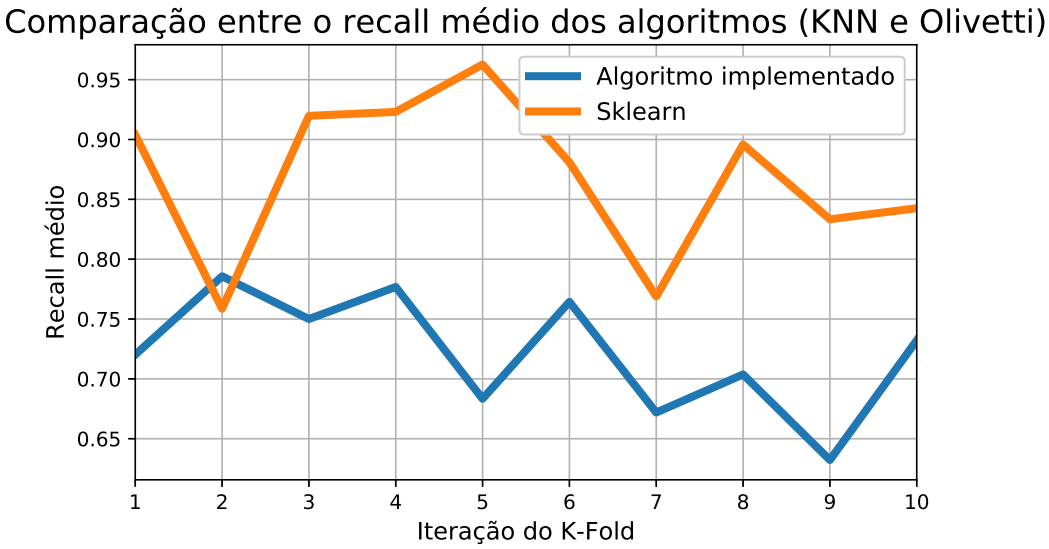

Figura 10. Recall médio obtido em cada iteração do método de validação cruzada K-Fold. Elaborado pelos autores.

Até o momento, os resultados obtidos apontam uma inadequação da implementação. Dessa forma, a equipe buscou analisar o algoritmo com outro *dataset*.

3.1.2. Iris Dataset

O próximo *dataset* utilizado para aferir o classificador KNN consiste em um conjunto de 50 amostras de dados. Cada dado é um vetor de quatro características que representa uma flor de iris qualquer, podendo ser classificado entre as classes *setosa*, *versicolour* e *virginica* [Fisher 1936].

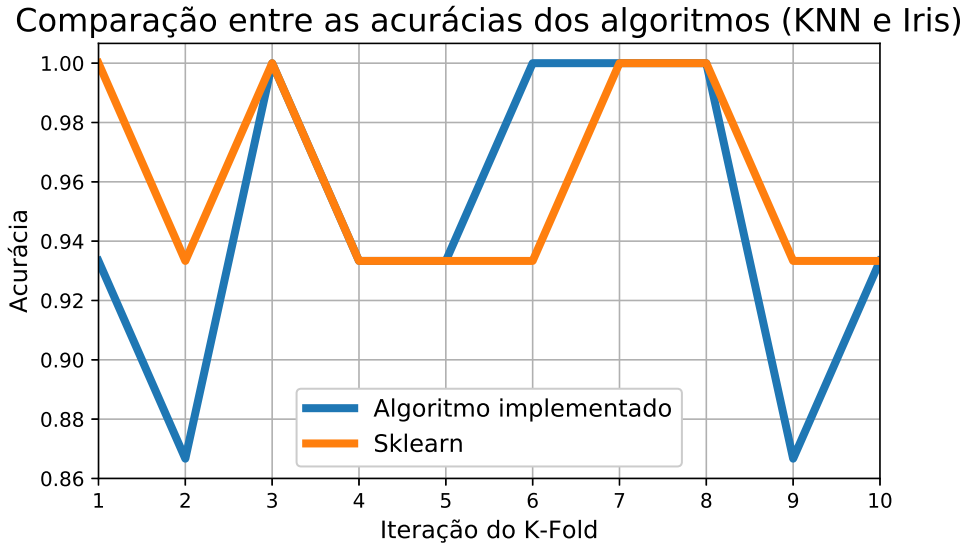

Figura 11. Acurácia obtida em cada iteração do método de validação cruzada K-Fold. Elaborado pelos autores.

A figura 11, acima, mostra uma melhoria satisfatória da implementação do KNN, com resultados próximos aos obtidos pela elaboração do *Scikit Learn*, quando considerada a métrica de acurácia.

Abaixo, sucedem-se os gráficos relativos às métricas *F1 Score* médio, precisão média e *Recall* médio para cada iteração do *K-Fold*, respectivamente.

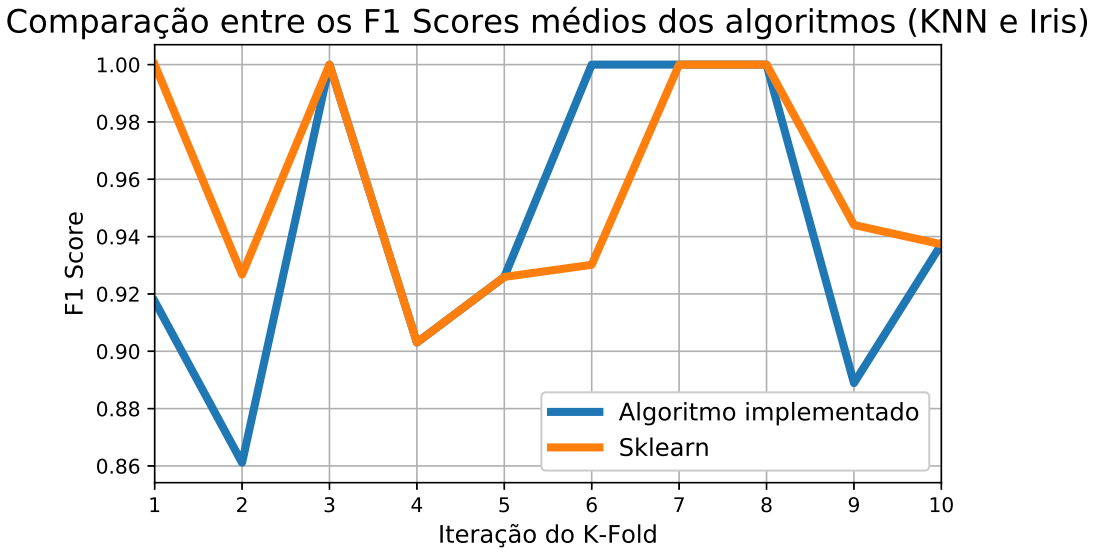

Figura 12. Média do F1 Score obtida em cada iteração do método de validação cruzada K-Fold. Elaborado pelos autores.

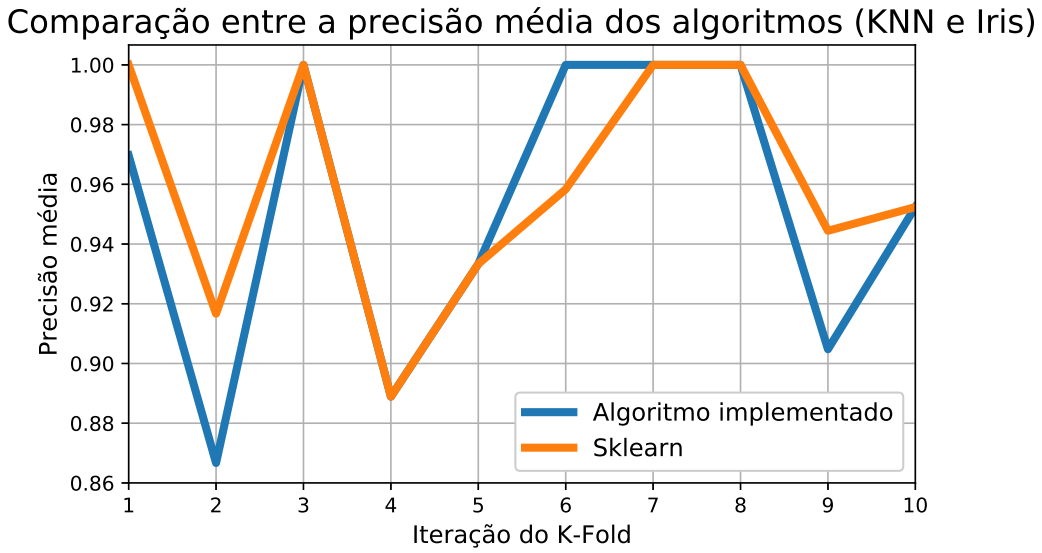

Figura 13. Precisão média obtida em cada iteração do método de validação cruzada K-Fold. Elaborado pelos autores.

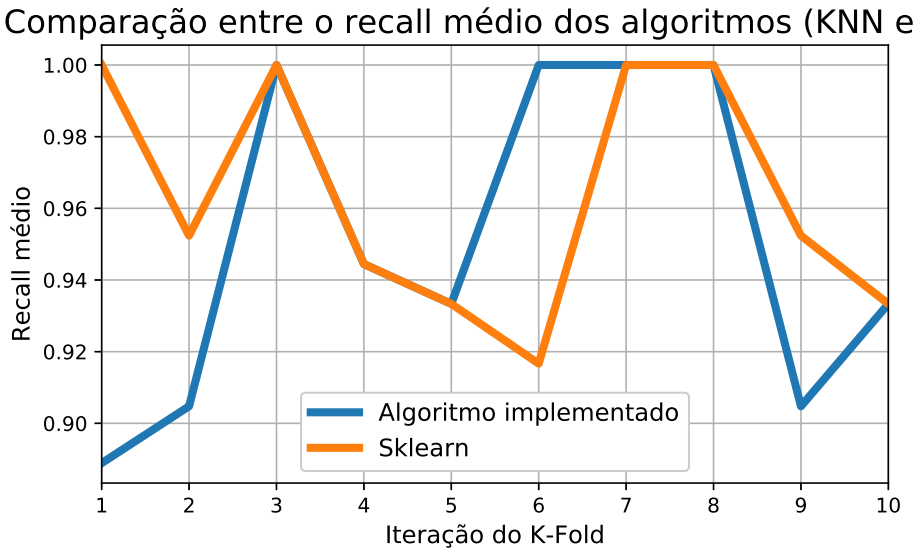

Figura 14. Recall médio obtido em cada iteração do método de validação cruzada K-Fold. Elaborado pelos autores.

A tabela da figura 15, abaixo, sintetiza os resultados obtidos nesta avaliação.

	KNN				
	Média das métricas utilizadas com K-Fold Cross Validation				
	Acurácia	F1 Score	Precisão	Recall	Dataset
Implementado	0,780	0,696	0,702	0,724	Olivetti Faces
Sklearn	0,870	0,826	0,831	0,840	
Implementado	0,960	0,960	0,966	0,964	Load_Iris
Sklearn	0,960	0,959	0,959	0,963	

Figura 15. Média de cada métrica utilizada no K-Fold Cross Validation para cada *dataset*. Elaborado pelos autores.

De acordo com a mesma tabela, em cada um dos *datasets* utilizados, a implementação do KNN obteve métricas menores ou ligeiramente superiores às das funções da biblioteca *Scikit Learn*. Segundo [Smola and Vishwanathan 2008], métodos não-paramétricos devem ser estudados com cautela, uma vez que o modelo usado pode cair em um problema conhecido como *curse of dimensionality*. De acordo com de acordo com a mesma fonte, no caso da norma Euclidiana, *features* com distribuições em diferentes escalas devem ser normalizadas de modo a possuir a mesma variância. Estes fatores podem levar o classificador a cometer erros que podem diminuir sua performance.

Baseado na suposição de que o modelo entrou de fato no problema da alta dimensionalidade dos dados, realizou-se a mesma análise a partir de uma base de dados de menor dimensionalidade. Neste caso, como pôde ser visto na tabela, houve uma melhoria considerável dos resultados ao utilizar o *"Iris Dataset"*. Por outro lado, verificando os resultados obtidos na análise do *dataset Olivetti Faces*, o algoritmo do *Scikit Learn* ainda manteve um desempenho alto. Supondo que este foi o caso do problema relacionado à diferença de escalas de distribuições, realizou-se uma análise das funções de classificação deste estudo e verificou-se que, ao contrário da função implementada pela equipe, o classificador do *Scikit Learn* possui em seu código uma seção tratativa para casos em que se

faz necessário a normalização dos dados de predição.

3.2. Perceptron

A fim de analisar o desempenho dos algoritmos do *Perceptron*, deve-se eleger um *dataset* de classificação binária, já que a saída desse algoritmo assume, necessariamente, os valores 0 ou 1. Nesse sentido, a equipe optou pelo *Breast Cancer Wisconsin Dataset*, constituído por 569 instâncias de 30 atributos numéricos pertencentes às classes *maligno*, 0, ou *benigno*, 1.

Por se tratar de um algoritmo classificador, as mesmas métricas empregadas na avaliação do KNN serão aplicadas às considerações sobre o *Perceptron*, i.e, acurácia, *F1 Score* médio, precisão média e *recall* médio. Assim como a implementação precedente, o *Perceptron* será explorado com auxílio do método de validação cruzada *K-Fold*.

Abaixo, a figura 16 mostra os valores de acurácia obtidos pelos algoritmos em cada iteração do *K-Fold Cross Validation*.

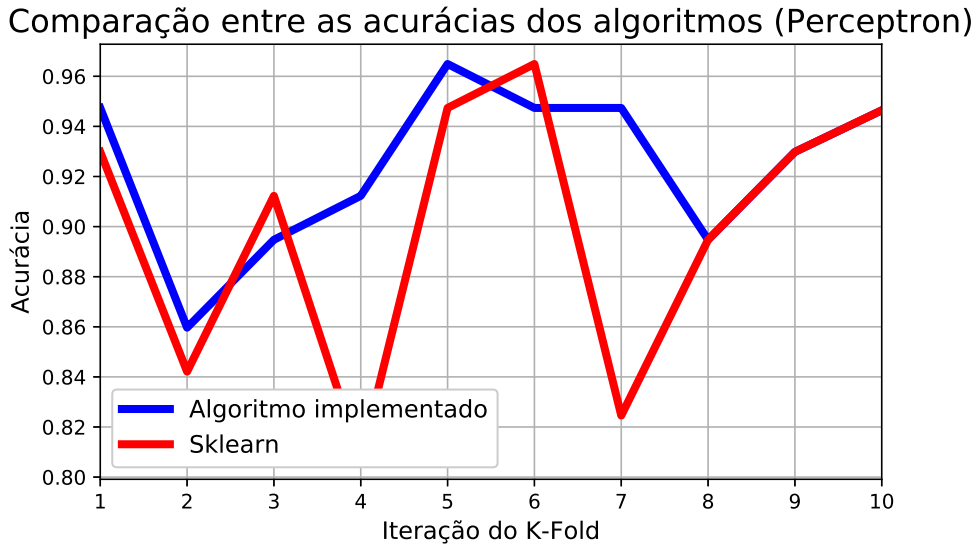

Figura 16. Acurácia obtida em cada iteração do método de validação cruzada K-Fold. Elaborado pelos autores.

Como é possível observar, a acurácia do algoritmo implementado pela equipe manteve-se, em boa parte das iterações, acima da acurácia do classificador do *Scikit Learn*. O mesmo padrão é observado na figura 17, no qual os valores de *F1 Score* médio permanecem maiores na maioria das iterações.

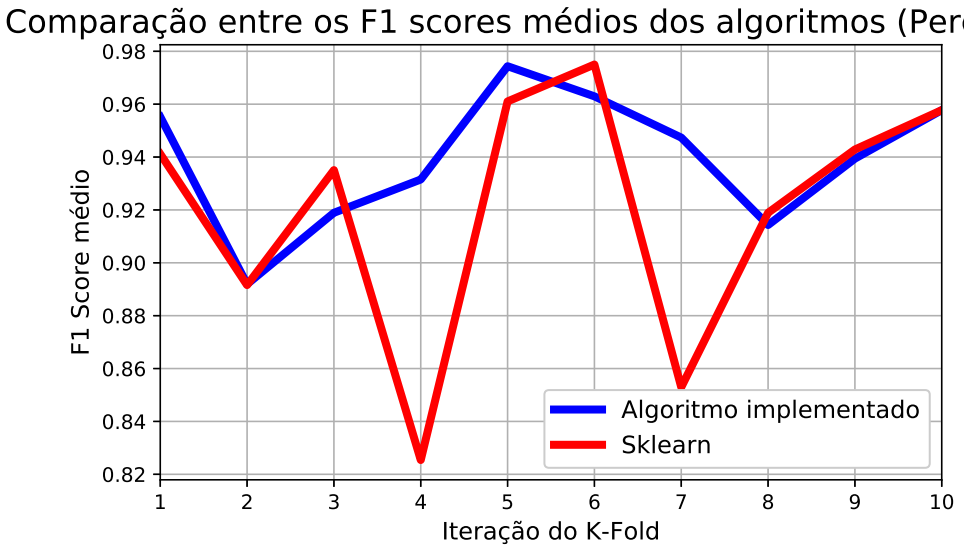

Figura 17. Média do F1 Score obtida em cada iteração do método de validação cruzada K-Fold. Elaborado pelos autores.

A figura 18, abaixo, segue a mesma orientação.

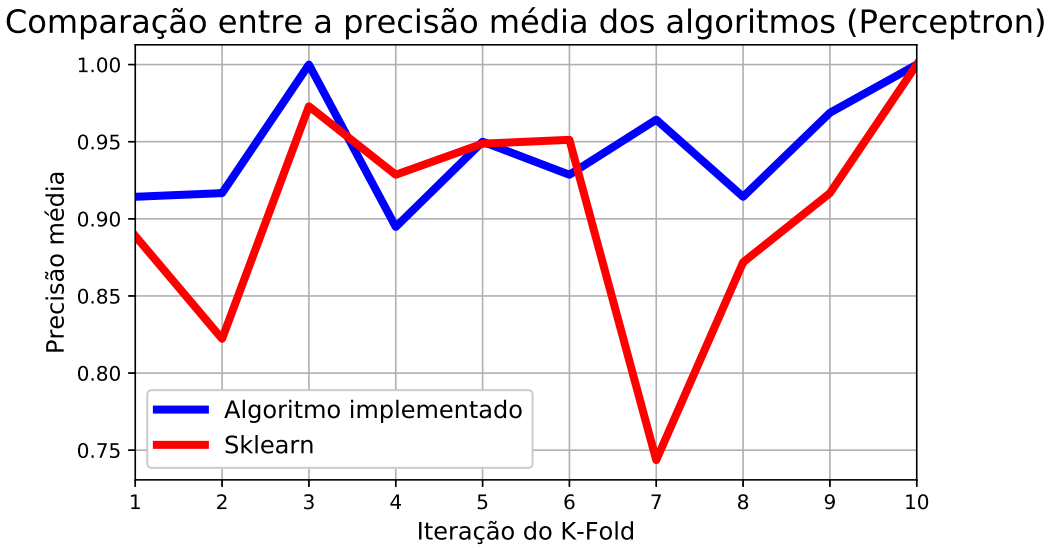

Figura 18. Precisão média obtida em cada iteração do método de validação cruzada K-Fold. Elaborado pelos autores.

A última métrica avaliada, *recall* médio, foi a única a quebrar o paradigma antes estabelecido. Na maioria das iterações, o algoritmo que se sobressai é, desta vez, o pertencente à biblioteca *Scikit Learn*.

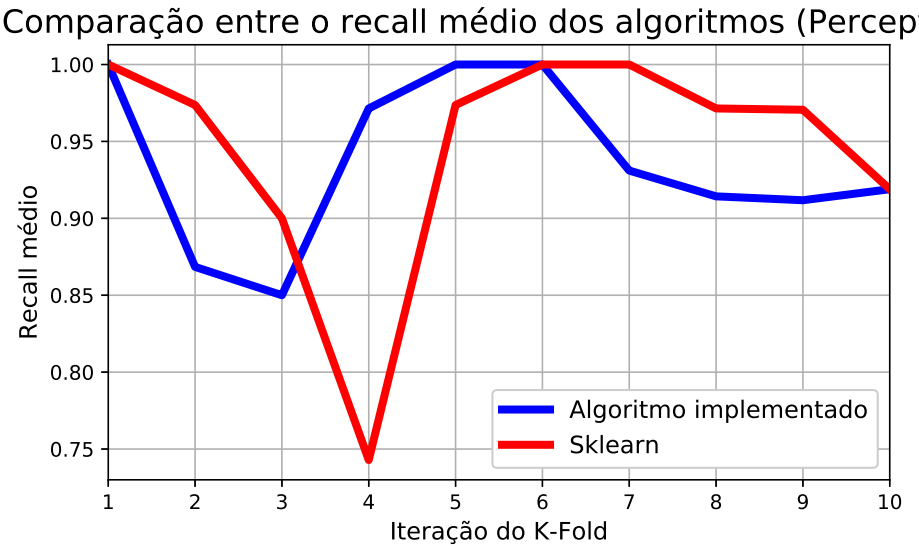

Figura 19. Recall médio obtido em cada iteração do método de validação cruzada K-Fold. Elaborado pelos autores.

A tabela da figura 20, abaixo, sumariza os valores médios obtidos para cada métrica. Nela, é possível confirmar que a única mensuração em que o algoritmo da biblioteca *Scikit Learn* se ressalta é o *recall* médio.

	PERCEPTRON Média das métricas utilizadas com K-Fold Cross Validation			
	Acurácia	F1 Score	Precisão	Recall
Implementado	0,905	0,924	0,947	0,909
Sklearn	0,903	0,920	0,927	0,924

Figura 20. Média de cada métrica utilizada no K-Fold Cross Validation. Elaborado pelos autores.

3.3. K-Means

Para examinar a qualidade dos agrupamentos produzidos pela implementação da equipe em comparação com o algoritmo já disponível na biblioteca *Scikit Learn*, utilizaram-se as métricas Coeficiente de Silhueta [Rousseeuw 1987], *Calinski & Harabaz Index* (também conhecido por *Variance Ratio Criterion*) [Caliński and Harabasz 1974] e o já detalhado *WB Index* [Zhao 2012]. Além disso, a equipe utilizou o método *K-Fold* de validação cruzada com dez pacotes, assim como na avaliação dos algoritmos abordados anteriormente nesse projeto.

O *dataset* utilizado pela equipe no treinamento desse algoritmo, nomeado *Wine Recognition Dataset*, constitui-se de treze atributos numéricos que caracterizam 178 elementos (vinhos). Apesar de apresentarem uma classificação autêntica, é possível aplicar esse *dataset* a um algoritmo de aprendizado não-supervisionado, como é o caso do *K-Means*.

A primeira métrica utilizada, Coeficiente de Silhueta, calcula a qualidade de um agrupamento de acordo com a proximidade de cada dado em um mesmo cluster juntamente com o afastamento dos clusters. O valor desse coeficiente pode ir de -1 a $+1$,

sendo ótimo o maior valor possível. A biblioteca *Scikit Learn* possui uma função de Coeficiente de Silhueta que foi usada para auferir os resultados a seguir:

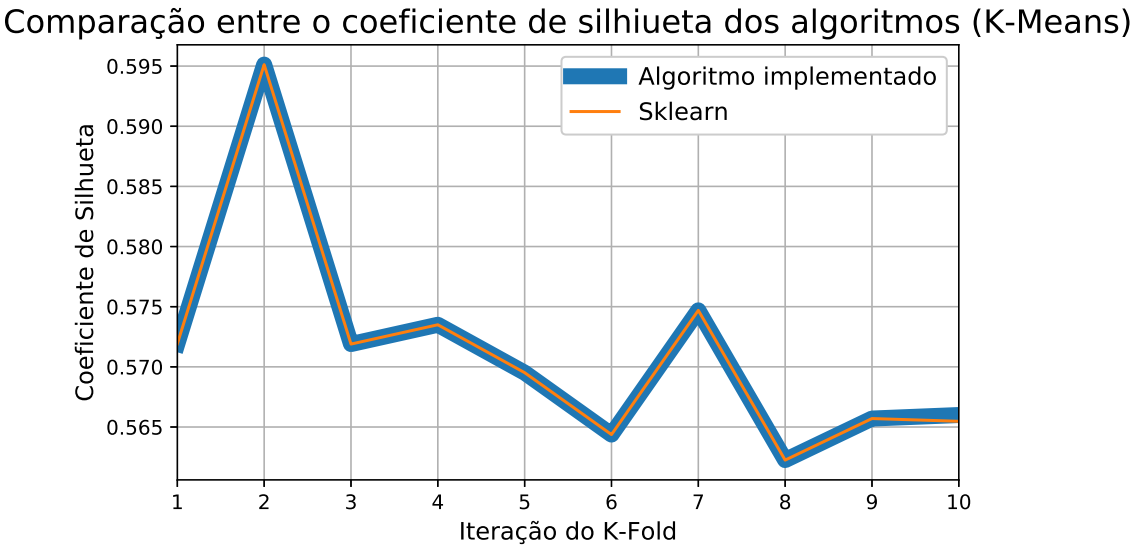

Figura 21. Comparação entre o coeficiente de silhueta dos agrupamentos produzidos pelos algoritmos em cada iteração do K-Fold Cross Validation. Elaborado pelos autores.

A figura 21 mostra os coeficientes obtidos com o algoritmo implementado pela equipe (em azul), extraordinariamente idênticos aos resultados conseguidos com o *K-Means* da biblioteca *Scikit Learn* (em laranja).

Em seguida, a equipe utilizou uma métrica conhecida por *Variance Ratio Criterion*. Essa mensuração é atribuída aos trabalhos de Calinski e Harabaz e, por isso, sua implementação na biblioteca *Scikit Learn* foi nomeada *Calinski & Harabaz Index*. Assim como o Coeficiente de Silhueta e o *WB Index*, o *Variance Ratio Criterion* também lida com a dispersão dos dados dentro dos agrupamentos e o espalhamento entre os *clusters*. Essa métrica assume valores mais altos conforme a separação e a densidade dos *clusters* aumenta [Caliński and Harabasz 1974] [Pedregosa et al. 2011].

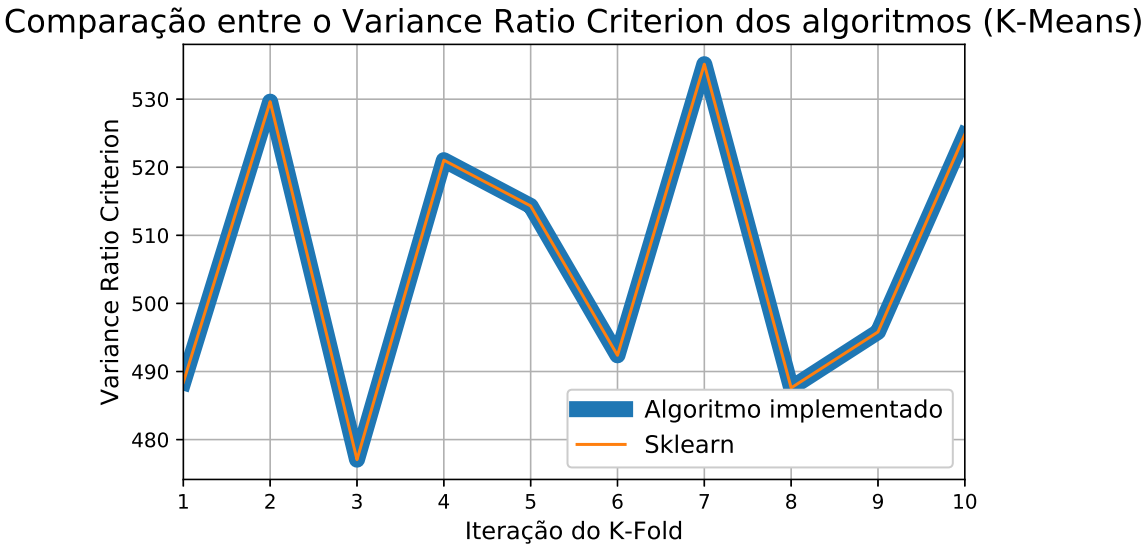

Figura 22. Comparação entre o Variance Ratio Criterion dos agrupamentos produzidos pelos algoritmos em cada iteração do K-Fold Cross Validation. Elaborado pelos autores.

Do mesmo modo que o Coeficiente de Silhueta, essa métrica também obteve os mesmos valores a cada iteração do *K-Fold Cross Validation* nos dois algoritmos avaliados, como mostra a figura 22.

Por fim, utilizou-se a métrica *WB Index* [Zhao 2012]. Como já mencionado previamente, seu algoritmo teve de ser implementado desde o inicio, já que a biblioteca *Scikit Learn* não o continha.

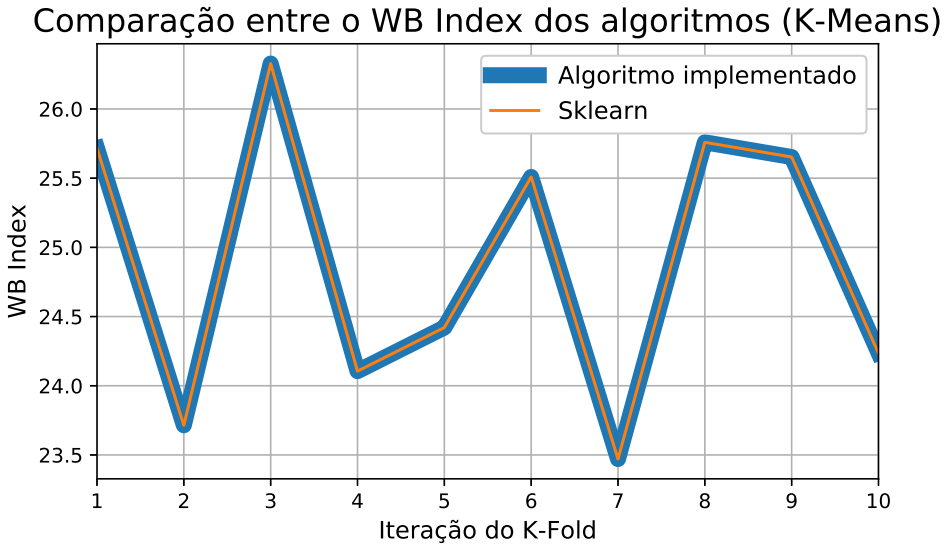

Figura 23. Comparação entre o WB Index dos agrupamentos produzidos pelos algoritmos em cada iteração do K-Fold Cross Validation. Elaborado pelos autores.

A última métrica empregada na avaliação dos algoritmos *K-Means* confirma que

a elaboração posta em prática pela equipe obteve um desempenho excelente, já que produziu resultados pariformes com o código implementado na biblioteca *Scikit Learn*.

A figura 24 mostra a média das métricas obtidas em cada iteração do *K-Fold Cross Validation*.

	K-MEANS		
	Média das métricas utilizadas com K-Fold Cross Validation		
	WB Index	Coef. Silhueta	Calinski & Harabaz Index
Implementado	24,891	0,571	506,612
Sklearn	24,891	0,571	506,622

Figura 24. Média de cada métrica utilizada no K-Fold Cross Validation. Elaborado pelos autores.

4. Considerações Finais

Durante o desenvolvimento desse trabalho, diversos tópicos referentes ao estudo de algoritmos de *Machine Learning* foram abordados.

Em um primeiro momento, a equipe buscou analisar a bibliografia para definir quais seriam os algoritmos implementados. Posteriormente, o grupo estudou os modelos já existentes na biblioteca *Scikit Learn* para melhor compreendê-los, possibilitando agregar o conhecimento necessário para explicar e elaborar as implementações pretendidas.

Em seguida, depois de implementar os algoritmos, a equipe desfrutou de uma série de métricas capazes de auxiliar a avaliação do desempenho dos modelos implementados, bem como compará-los com os códigos da biblioteca *Scikit Learn*. Tais métricas utilizadas incluem a acurácia, *F1 Score*, precisão e *recall* para os algoritmos classificadores, bem como coeficiente de silhueta e *Calinski & Harabaz Index* para os clusterizadores. Além disso, o grupo ainda implementou uma métrica ausente na biblioteca de referência, denominada *WB Index*.

Com as métricas referidas, foi possível colocar à prova os algoritmos implementados e os da biblioteca *Scikit Learn*. Inicialmente, os testes com KNN revelaram uma péssima resposta do classificador para a primeira base de dados, entretanto, considerando as possíveis causas quanto à distribuição das características das amostras, verificou-se alguns a necessidade de algumas implementações para a melhoria dos resultados, as causas puderam ser conferidas testando o algoritmo em uma base de dados de menor dimensionalidade. Já na segunda fase de testes, dessa vez usando o *Perceptron*, as métricas, por diversas vezes, revelaram seus melhores valores quando utilizado o algoritmo implementado pela equipe, superando as expectativas dos integrantes. Por último, os resultados obtidos com a implementação do K-Means foram extraordinários por apresentarem exatamente os mesmos valores conseguidos com a biblioteca de referência, sugerindo que ambas as implementações podem ser muito parecidas.

Além disso, com o surgimento de problemas relacionados à influência da forma e das características estatísticas do *dataset* nos resultados, constatou-se a importância de trabalhar as amostras de um conjunto de dados para que não houvesse uma diminuição da performance do modelo quanto às métricas, ou, ainda, para que não ocorra um engano na interpretação dos resultados preditos por um modelo.

De uma maneira geral, a equipe acredita que os objetivos almejados no inicio do projeto foram alcançados com êxito, proporcionando um aprendizado mais profundo acerca de algoritmos de *Machine Learning*.

Referências

Alpaydin, E. (2009). *Introduction to Machine Learning (Adaptive Computation and Machine Learning series)*. The MIT Press.

Altman, N. S. (1992). An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185.

Borodin, A. and El-Yaniv, R. (1998). *Online Computation and Competitive Analysis*. Cambridge University Press.

Caliński, T. and Harabasz, J. (1974). A dendrite method for cluster analysis. *Communications in Statistics*, 3(1):1–27.

Cambridge University Engineering Department (1992). *Olivetti Faces Dataset*. AT&T Laboratories Cambridge.

Data Science Academy (2018). *Deep Learning Book*. Data Science Academy.

Fisher, R. (1936). *The Use of Multiple Measurements in Taxonomic Problems*. Annual Eugenics, 7, Part II, 179-188.

Lichman, M. (2013). *UCI Machine Learning Repository*. University of California, School of Information and Computer Science.

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. In *Proc. Fifth Berkeley Sympos. Math. Statist. and Probability (Berkeley, Calif., 1965/66)*, pages Vol. I: Statistics, pp. 281–297. Univ. California Press, Berkeley, Calif.

Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective (Adaptive Computation and Machine Learning series)*. The MIT Press.

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

Rosenblatt, F. (1957). The perceptron – a perceiving and recognizing automaton. *Cornell Aeronautical Laboratory*, 85-460-1.

Rousseeuw, P. J. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53 – 65.

Shalev-Shwartz, S. and Ben-David, S. (2014). *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press.

Smola, A. and Vishwanathan, S. (2008). *Introduction to Machine Learning*. Cambridge University Press.

W.N. Street, W. W. and Mangasarian, O. (1993). *Nuclear Feature Extraction for Breast Tumor Diagnosis*. International Symposium on Electronic Imaging: Science and Technology.

Zhao, Q. (2012). *Cluster Validity in Clustering Methods*. University of Eastern Finland.