UNIVERSIDADE FEDERAL DO ABC

Tópicos em Inteligência Artificial: Machine Learning

IMPLEMENTAÇÃO E ANÁLISE DO ALGORITMO K NEAREST NEIGHBORS

Dimitri Leandro de Oliveira Silva

Prof. Dr. André Kazuo
Prof. Dr. Ricardo Suyama

Santo André
Março de 2020

Resumo

O presente documento trata da implementação do algoritmo conhecido como K Vizinhos Mais Próximos (KNN). A linguagem de programação utilizada foi Python, já que, assim, seria possível fazer comparações com as soluções já disponibilizadas pela biblioteca Scikit Learn. Foram criadas duas classes, uma para o KNN e outra para realizar o cálculo das métricas de acurácia, precisão e revocação. O algoritmo de ordenação implementado foi o Selection Sort, devido à sua simplicidade. Diversos datasets foram testados. Os resultados mostram que o algoritmo implementado obteve resultados idênticos ao da solução do Scikit Learn, mas que, apesar disso, apresentou um tempo de processamento muito mais demorado. Além disso, observou-se que o KNN apresentou um resultado muito satisfatório para datasets sem sobreposição de classes, mesmo que elas não fossem linearmente separáveis. Os datasets foram testados para vários valores de K (vizinhos mais próximos), onde, para cada K, utilizou-se Bootstrap como técnica de validação cruzada.

Sumário

1 Introdução

O algoritmo conhecido como K Vizinhos Mais Próximos (K Nearest Neighbors ou KNN) é um algoritmo de classificação no ramo de aprendizado de máquina capaz de fazer previsões relacionando um dado desconhecido à uma determinada categoria. O KNN é um dos algoritmos mais simples utilizados para classificação e regressão, baseando-se na distância entre um novo dado que se deseja classificar e dados previamente conhecidos. Dessa forma, ele computa a distância entre um novo dado e todos os outros já armazenados para predizer a classificação deste dado desconhecido, utilizando a moda das classificações dos K dados mais próximos no espaço vetorial de features para isso [1, 2, 3].

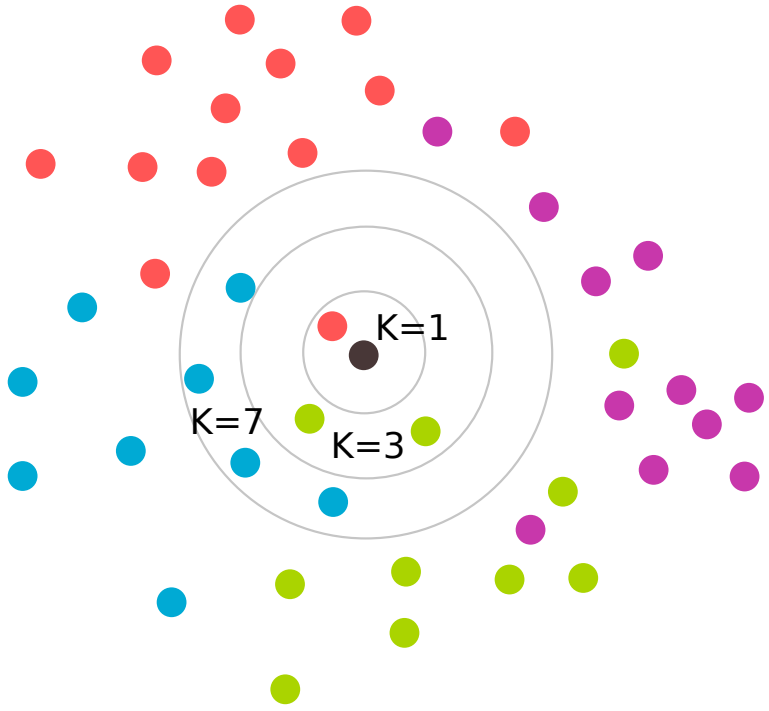

Figura 1: Exemplo de funcionamento do algoritmo KNN.

Fonte: Elaborado pelo autor.

A Figura 1 exemplifica o algoritmo do KNN. Nela, mostra-se um conjunto de dados com classificação já conhecida num espaço de características bidimensional. Ao se inserir um dado com classificação desconhecida no espaço vetorial, pode-se definir um valor de K para que seja feita a predição da classificação deste novo dado. Como é possível perceber, caso K seja igual a um, o dado será categorizado na classe vermelha. Caso K seja igual a três, a classificação passa a ser da classe verde e, caso

K seja igual a sete, a classificação muda novamente, desta vez, para azul [4].

O KNN é um algoritmo não paramétrico, isto é, não possui um número definido de parâmetros. Ele utilizará todos os dados armazenados para determinar a classificação de um dado desconhecido, diferentemente de algoritmos paramétricos, como o Perceptron, que utiliza apenas pesos previamente definidos para realizar a classificação. Isso faz com que sua etapa de treinamento seja extremamente rápida, bastando apenas armazenar os dados selecionados para treinamento, mas implica numa lentidão considerável para classificar novos dados [2, 3].

O objetivo deste trabalho é implementar este algoritmo e compará-lo com a solução já disponibilizada pela biblioteca de Python, Scikit Learn [5].

2 Metodologia

Este trabalho utilizou uma série de datasets para testar o algoritmo implementado e compará-lo com o já disponível na biblioteca Scikit Learn. Em todos eles, foram feitas comparações de acurácia, precisão média, revocação média e tempo de processamento.

Os testes foram divididos em três partes. Na primeira, foram utilizados datasets bidimensionais simples com uma quantidade de dados pequena. Na segunda, foram novamente utilizados datasets bidimensionais, mas, desta vez, maiores e mais complexos. Por último, foi utilizado o dataset Olivetti Faces, um conjunto de dados contendo fotos do rosto de quarenta indivíduos, sendo dez fotos para cada pessoa. Cada foto continha um total de 4096 pixels que foram utilizados como features numa escala de cinza.

A Figura 2 mostra os datasets utilizados na primeira parte do projeto. E a Figura 3 exibe os datasets bidimensionais mais complexos, utilizados na segunda parte do projeto.

Para implementar o algoritmo, utilizou-se a linguagem de programação Python. O método adotado para calcular a distância entre dois pontos foi o euclidiano, mas é importante lembrar que outros tipos distâncias também poderiam ter sido utilizados, tais como Manhattan e Minkowski. Para ordenar os dados armazenados no treinamento da menor para a maior distância em relação à um novo dado, implementou-se o

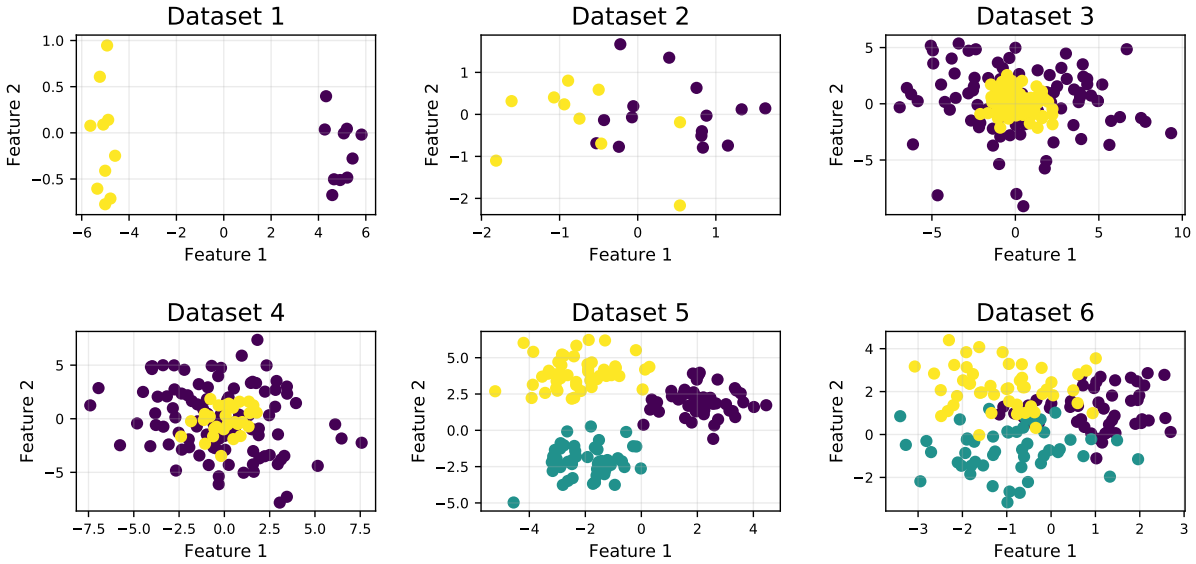

Figura 2: Datasets da primeira parte do projeto.

Fonte: Elaborado pelo autor.

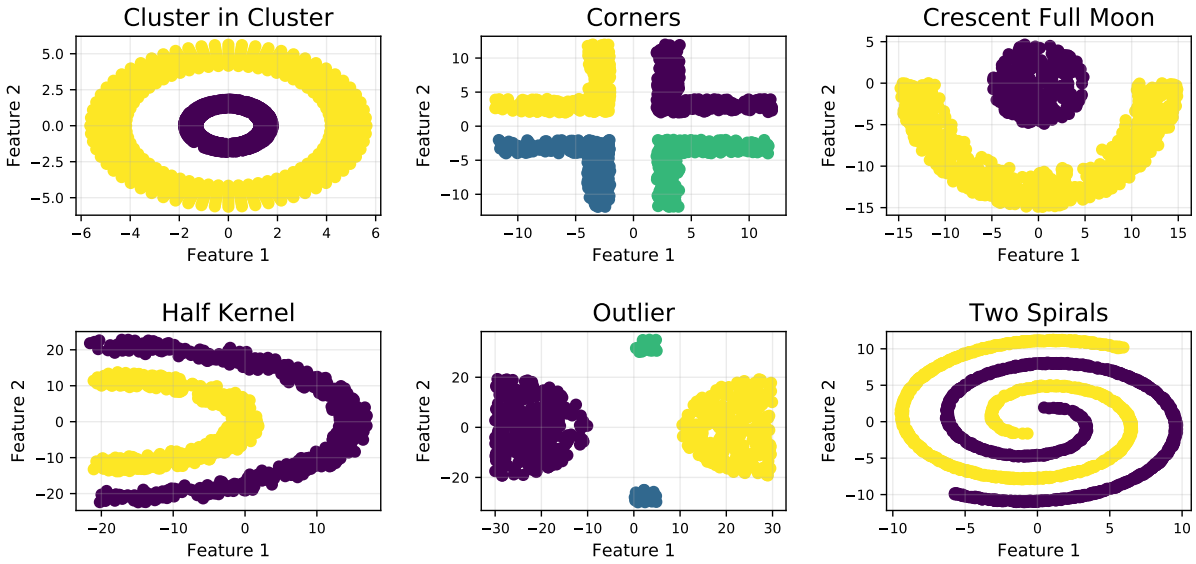

Figura 3: Datasets da primeira parte do projeto.

Fonte: Elaborado pelo autor.

algoritmo de ordenação Selection Sort, devido à sua simplicidade. Ainda sobre as implementações, construiu-se uma classe somente para realizar o cálculo das métricas de acurácia, precisão e revocação dado uma matriz de confusão[1].

A técnica de validação cruzada utilizada para dar mais confiabilidade aos resul-

[1]Todos os códigos implementados podem ser encontrados em https://github.com/DimitriLeandro/MachineLearningUFABC/tree/master/Projeto_KNN.

tados é denominada Bootstrap. Essa técnica consiste da utilização de vários Holdouts para que as métricas sejam avaliadas na média. O Holdout consiste na separação dos dados de um dataset entre treinamento e teste, onde é definido um percentual de dados que serão separados para cada funcionalidade. Depois disso, o classificador é treinado e as métricas são avaliadas. A Figura 4 esquematiza a metodologia utilizada para gerar os resultados em cada um dos datasets [3, 4].

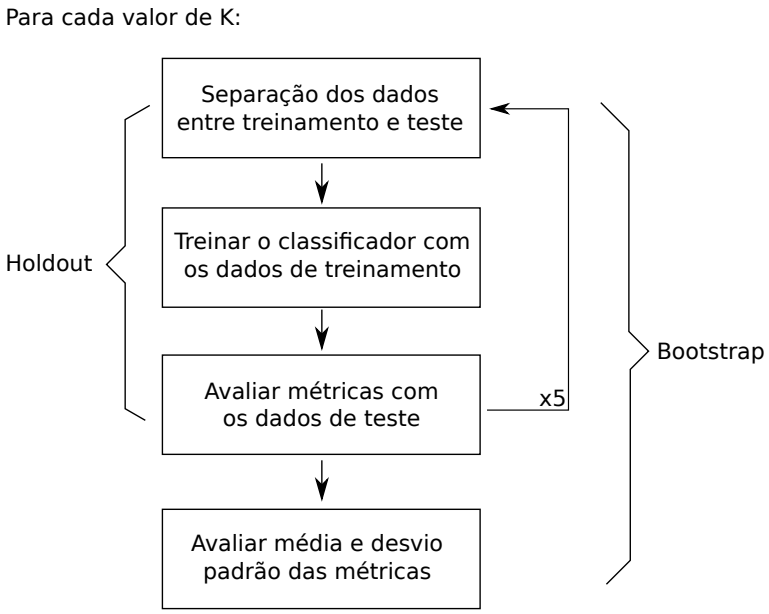

Figura 4: Metodologia utilizada para gerar os resultados.

Fonte: Elaborado pelo autor.

Nos datasets da primeira parte do projeto, variou-se o valor de K de um até nove, com passo dois. Já nas segunda e terceira partes, variou-se esse valor de um até dezenove. Para cada valor de K, foram feitas cinco iterações do Holdout, onde 75% dos dados foram separados aleatoriamente para treinamento e o restante para teste.

3 Resultados e Discussões

Esta seção mostrará os resultados obtidos para todos os datasets utilizados no projeto. Todos os gráficos apresentam o mesmo padrão: uma comparação entre o algoritmo implementado e o já disponível pela biblioteca Scikit Learn. Para isso, serão apresentadas as métricas de acurácia, precisão média, revocação média e tempo de

processamento. Entretanto, as classificações atribuídas à um novo dado, tanto pelo algoritmo implementado quanto pelo da biblioteca Scikit Learn, são idênticas. Logo, ao invés de mostrar oito retas diferentes, os gráficos mostrarão apenas cinco, já que as métricas de acurácia, precisão média e revocação média foram iguais para ambos os algoritmos utilizados.

3.1 Parte I

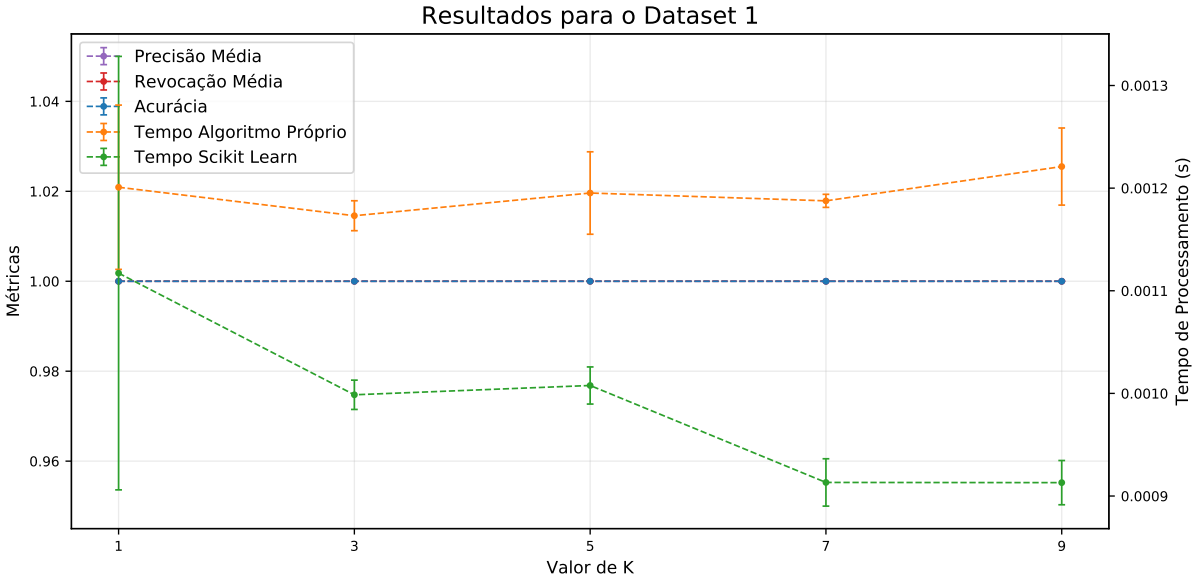

Figura 5: Resultados obtidos com o dataset 1.

Fonte: Elaborado pelo autor.

A Figura 5 mostra os resultados obtidos para o primeiro dataset da Parte I. Observa-se que a acurácia, precisão média e revocação média mantiveram-se em 100% para todos os valores de K testados. É possível explicar esse comportamento dado que claramente não há sobreposição entre as classes desse dataset. Por isso, o KNN não encontra dificuldades em determinar à qual classe pertence um novo dado. Além disso, percebe-se que o tempo médio para processar cada holdout em cada valor de K foi muito maior no algoritmo implementado do que no disponibilizado pelo Scikit Learn. Acredita-se que isso se deva já que essa biblioteca costuma paralelizar seus algoritmos, além de otimizá-los ao máximo. Caso o algoritmo implementado também tivesse sido paralelizado, ou caso o algoritmo do Scikit Learn fosse forçado a utilizar somente um único núcleo do processador, então os tempos de processamento dos

dois algoritmos aproximariam-se.

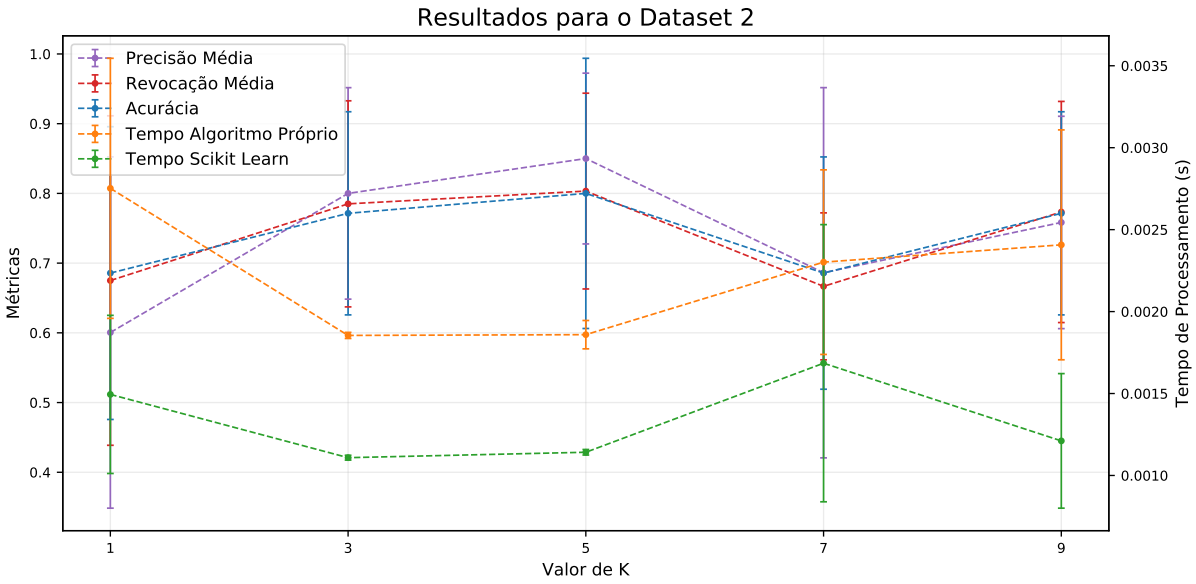

Figura 6: Resultados obtidos com o dataset 2.

Fonte: Elaborado pelo autor.

A Figura 6 mostra os resultados obtidos para o segundo dataset. Dessa vez, como há sobreposição entre as classes, as métricas caíram significativamente. O melhor modelo obtido foi para o caso em que K era igual a cinco, onde a média da acurácia e média das revocações médias ficaram em cerca de 80%. A precisão obteve um resultado ainda melhor.

A Figura 7 mostra os resultados para o terceiro dataset. Nota-se que em todos os casos o algoritmo implementado foi muito mais lento que o disponibilizado pelo biblioteca Scikit Learn.

A Figura 8 mostra os resultados para o quarto dataset. O melhor resultado em termos de acurácia foi para o caso em que K foi nove, onde essa métrica obteve média pouco abaixo de 80%. Entretanto, em termos de precisão média e revocação média, talvez o melhor resultado se traduza para o caso em que K foi igual a um. O melhor valor escolhido para K nem sempre vai depender apenas da acurácia, visto que para uma determinada aplicação, outras métricas podem ser mais significativas.

Os resultados para o quinto dataset, apresentados na Figura 9, chamam atenção para o caso em que K é igual a sete. Todas as métricas atingiram 100% e em todas as iterações do bootstrap, por isso nem apresentam barra de desvio padrão.

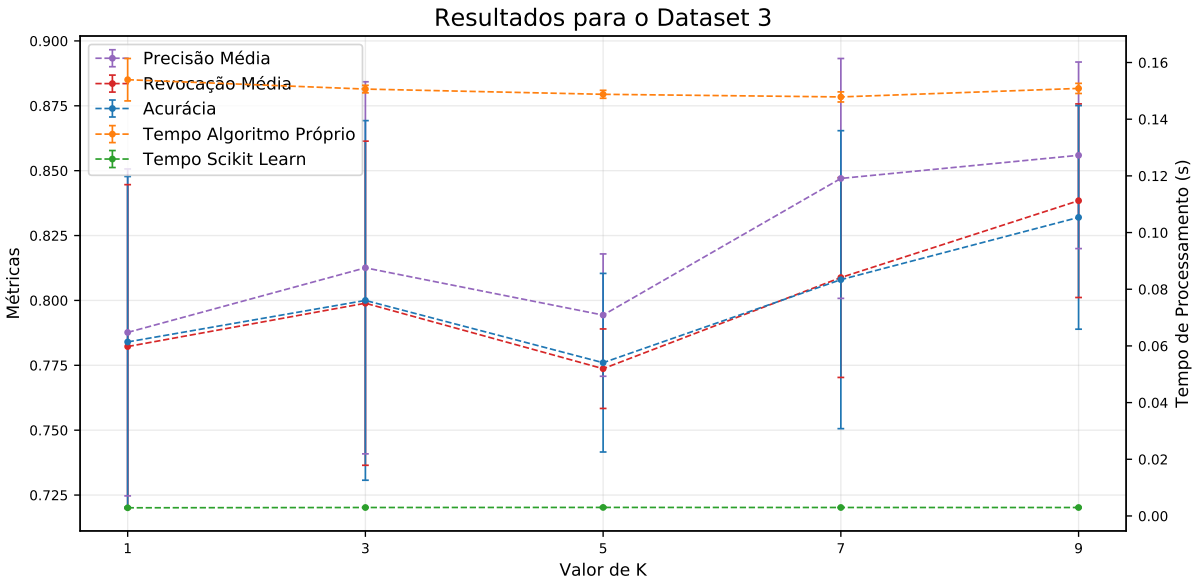

Figura 7: Resultados obtidos com o dataset 3.

Fonte: Elaborado pelo autor.

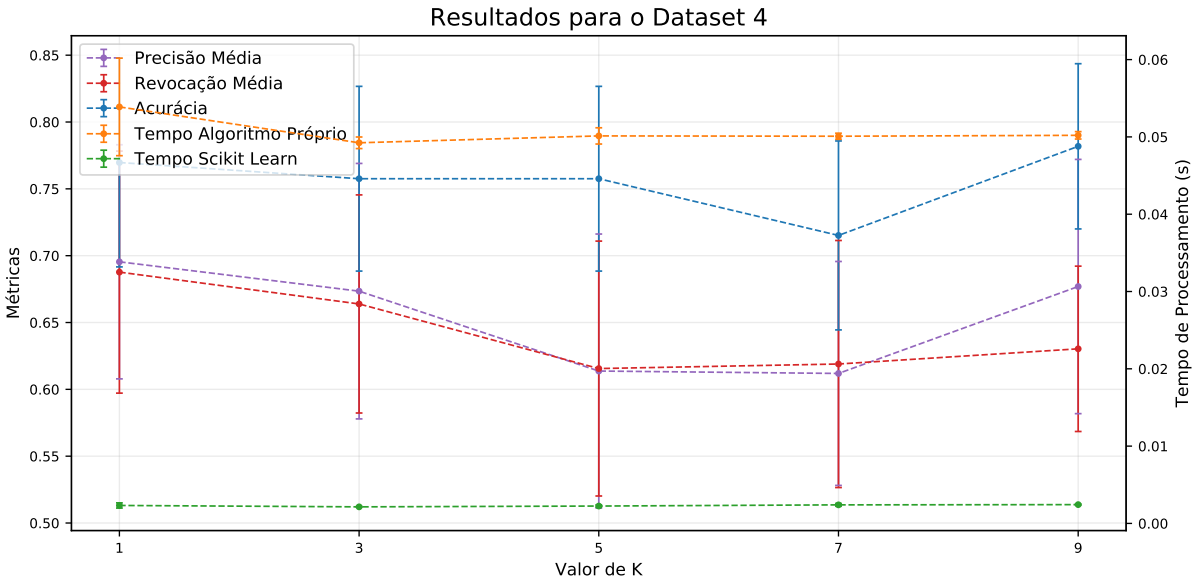

Figura 8: Resultados obtidos com o dataset 4.

Fonte: Elaborado pelo autor.

Esse, sem dúvida, é o valor ideal de K para esse dataset.

Por último, a Figura 10 mostra os resultados para o sexto dataset.

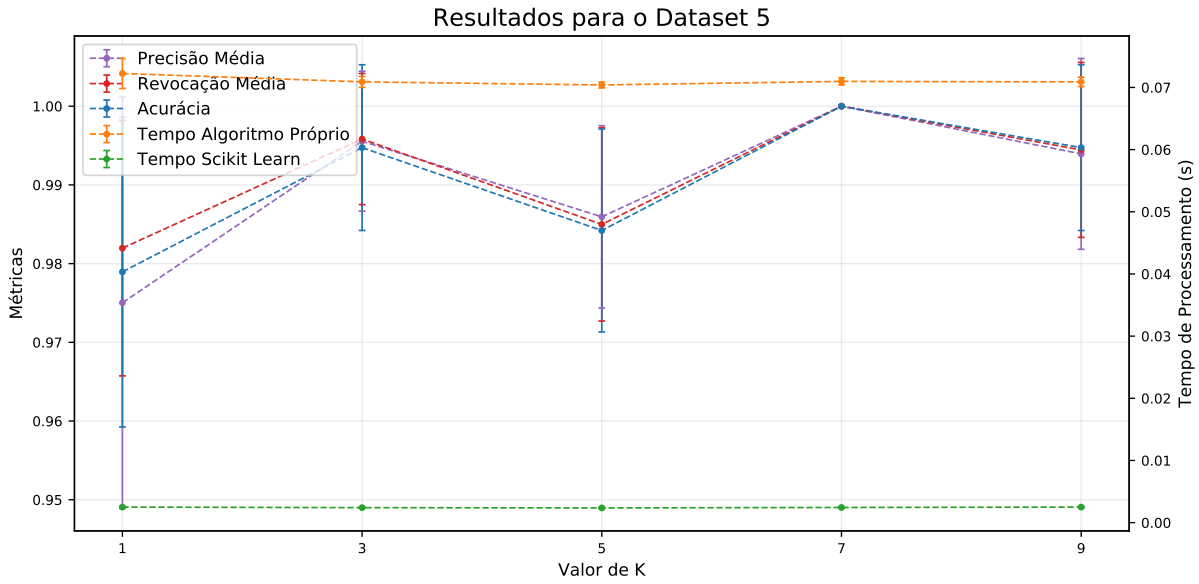

Figura 9: Resultados obtidos com o dataset 5.

Fonte: Elaborado pelo autor.

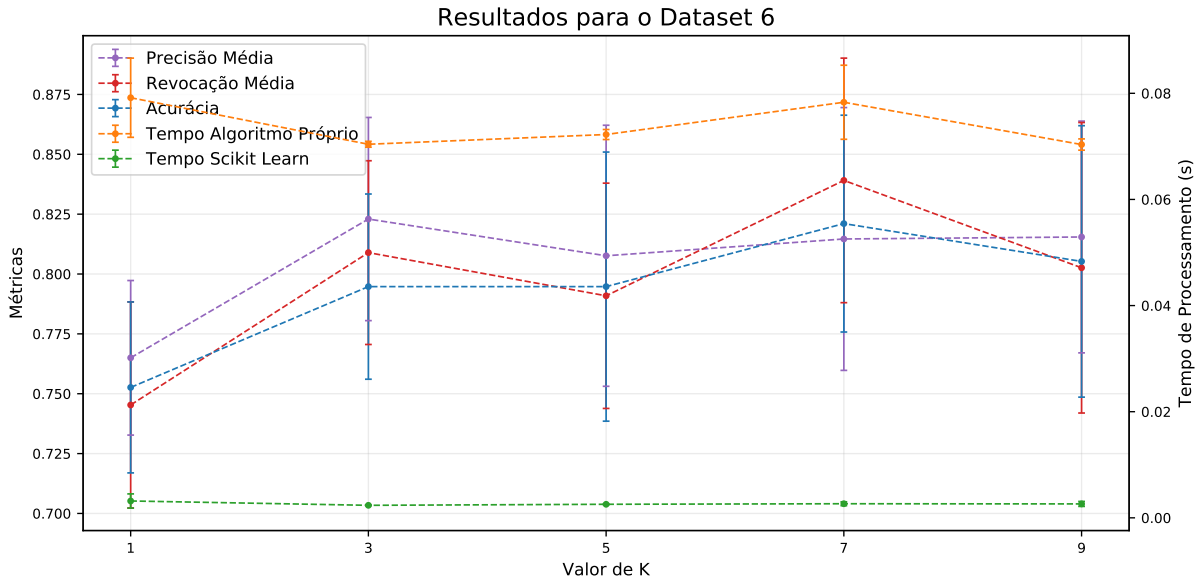

Figura 10: Resultados obtidos com o dataset 6.

Fonte: Elaborado pelo autor.

3.2 Parte II

A Figura 11 mostra os resultados obtidos para os datasets da segunda parte do projeto.

Todos os gráficos mostram o mesmo comportamento. As métricas de acurácia, precisão média e revocação média mantiveram-se no maior valor possível para todos

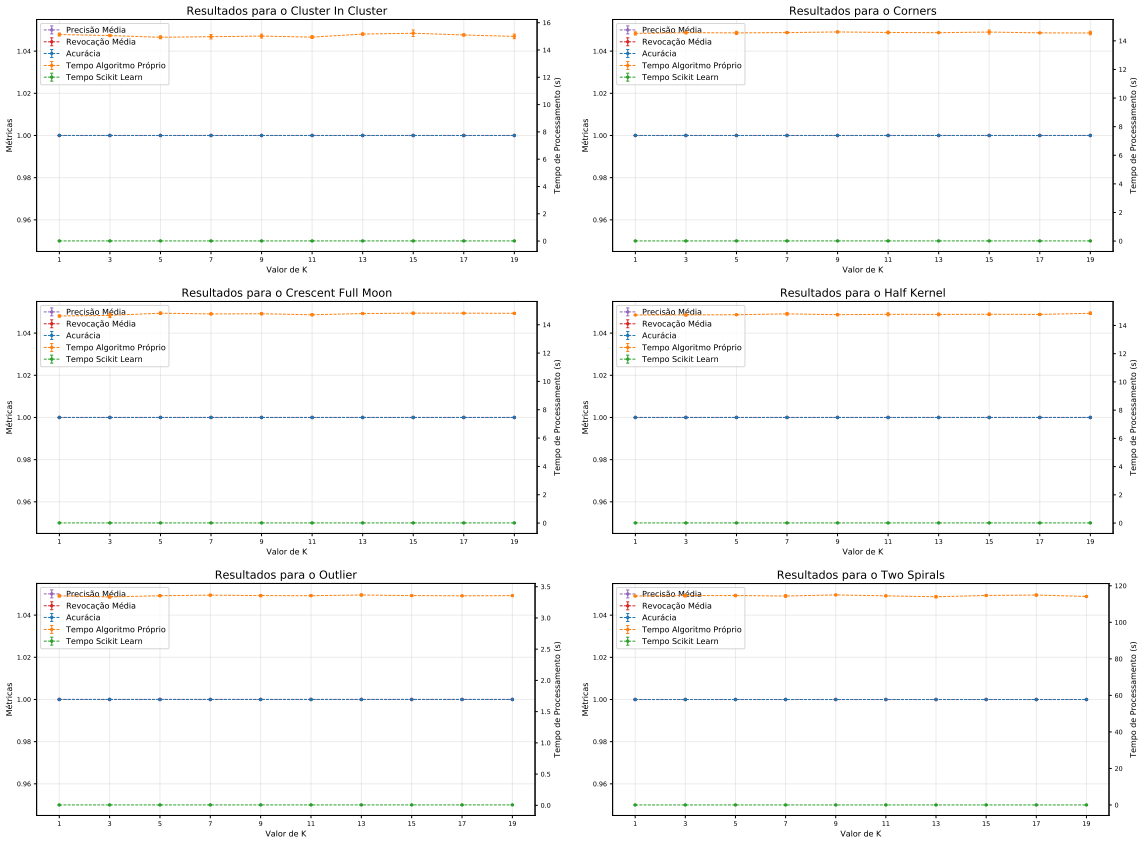

Figura 11: Resultados obtidos na segunda parte do projeto.

Fonte: Elaborado pelo autor.

os valores de K testados e em todas as iterações do bootstrap. Isso explica a ausência das barras de desvio padrão.

Pode-se explicar esse comportamento fazendo uma rápida análise visual dos datasets da Parte II, na Figura 3 da Seção 2. Todos os datasets estudados nessa parte não possuem sobreposição entre as classes. Isto é, observa-se nitidamente uma fronteira entre os pontos de cores diferentes, o que não acontece para os datasets da Parte I, por exemplo. Portanto, mesmo para valores muito altos de K, o algoritmo consegue classificar facilmente os pontos desses datasets.

É muito importante lembrar que, mesmo que os resultados tenham sido extremamente positivos para esses datasets quando usado o KNN, não necessariamente teriam sido tão bons caso fossem utilizados outros classificadores. É razoável supor que para classificadores lineares, tais como o Perceptron, SVM Linear, LDA e entre outros, os resultados não seriam tão satisfatórios. Basta verificar que, dado que os classificadores citados são lineares, então para espaços de características bidimensi-

onais como os da segunda parte, as fronteiras de decisão seriam retas. Observando a Figura 3, verifica-se que não seria possível encontrar uma reta capaz de separar satisfatoriamente as classes desses datasets e que, portanto, os resultados seriam prejudicados [3, 2, 4].

Sobre o tempo de processamento, mais uma vez o algoritmo implementado pelo Scikit Learn foi mais rápido. A média de tempo para cada iteração do Holdout em cada valor de K foi de pouco mais de 0 segundos, enquanto que, o algoritmo implementado levou cerca de 15 segundo em todos os datasets, exceto pelo dataset Outlier, que levou pouco mais de 3 segundos. Percebe-se que em todos os casos o desvio padrão dessa métrica foi muito baixo, o que indica uma alta confiabilidade nos resultados obtidos, tornando possível a realização de estimativas de tempo de processamento para futuros experimentos.

3.3 Parte III

A Figura 12 mostra os resultados obtidos para o dataset Olivetti Faces.

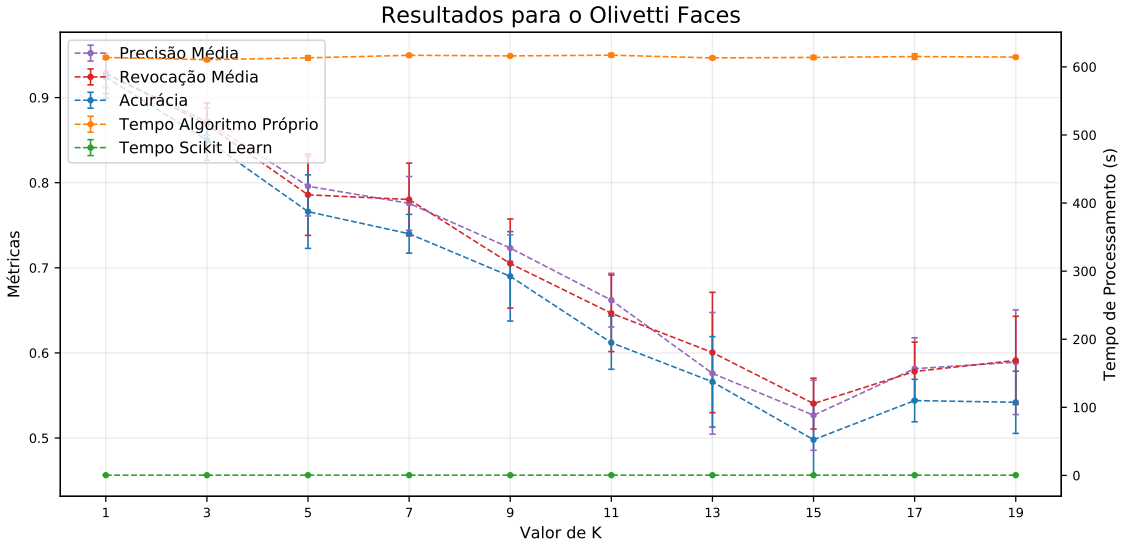

Figura 12: Resultados obtidos com o dataset Olivetti Faces.

Fonte: Elaborado pelo autor.

Observa-se que as principais métricas decaíram rapidamente conforme o valor de K aumentou. Quando o número de vizinhos mais próximos foi igual a um, obtiveram-se os melhores resultados de acurácia, precisão média e revocação média, todas maiores que 90%. Entretanto, conforme K aumentou, os resultados começaram a piorar.

A acurácia chegou a obter um valor médio de 50% quando o número de vizinhos mais próximos foi quinze. Ainda assim, esse valor é muito melhor que o de um classificador aleatório, que para o caso de quarenta classes, como é o desse dataset, teria obtido uma acurácia de apenas 2,5%.

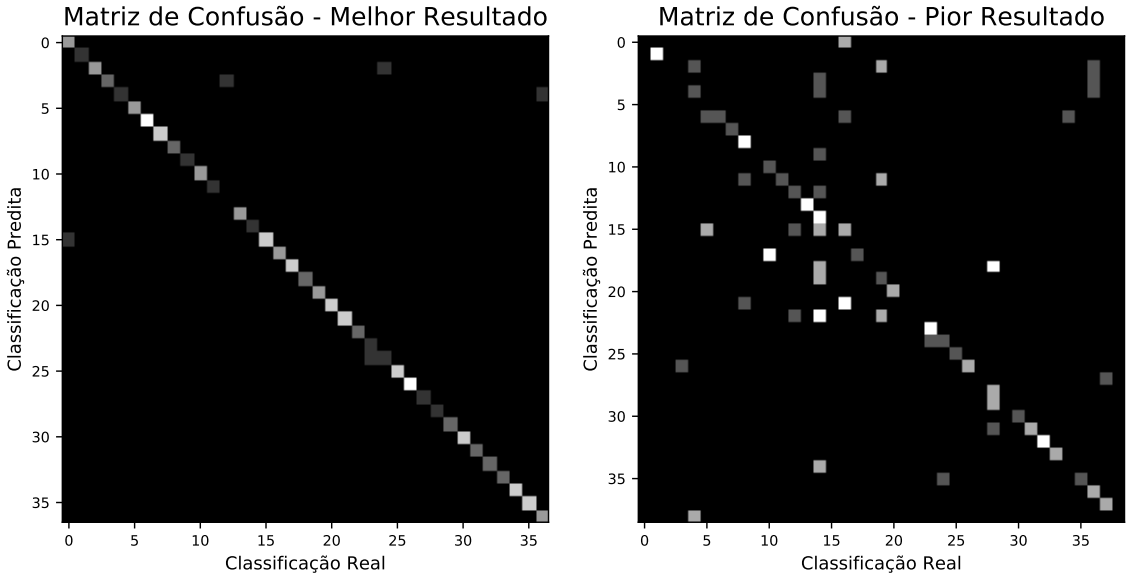

Figura 13: Matrizes de confusão obtidas nos testes com o Olivetti Faces. A figura da esquerda mostra o melhor resultado obtido em termos da acurácia, já a figura da esquerda mostra o pior resultado obtido.

Fonte: Elaborado pelo autor.

A Figura 13 mostra duas matrizes de confusão obtidas nos testes com o Olivetti Faces. À esquerda, verifica-se o melhor resultado obtido, quando K foi igual a um. Nessa matriz, é possível perceber que a diagonal principal apresenta cores muito claras e, ao redor, cores mais escuras. Isso indica que houve um alto índice de acertos. Algumas classes ainda conseguiram obter 100% de precisão e revocação, como é o caso das classes 6 e 26. Já a matriz à direita apresenta o pior resultado obtido, quando K foi igual a quinze. Nela, é possível perceber que o classificador cometeu muitos erros e, em alguns casos, errou todas as classificações referentes ao rosto de uma determinada pessoa, como foi o caso da classe 34, onde, de todas as vezes em que uma amostra dessa classe foi apresentada ao classificador, ele acreditou se tratar da classe 6.

4 Considerações Finais

O algoritmo KNN foi implementado com êxito, tendo obtido exatamente os mesmos resultados que os do algoritmo disponibilizado pelo Scikit Learn. Entretanto, o algoritmo implementado apresentou um tempo de processamento muito maior. Acredita-se que isso tenha ocorrido já que o Scikit Learn paraleliza seus algoritmos, fazendo com que vários núcleos do processador trabalhem para obter os resultados mais rapidamente. Além disso, o algoritmo de ordenação implementado, Selection Sort, costuma ser consideravelmente mais lento que outras alternativas, como o Quicksort. Apesar disso, os gráficos mostram que o tempo de processamento do algoritmo implementado, apesar de alto, variou muito pouco ao redor da média.

Além disso, percebe-se que o KNN trabalha muito bem com datasets sem sobreposição entre as classes, mesmo que elas não sejam linearmente separáveis. Como visto, o algoritmo obteve 100% nas métricas testadas para todos os valores de K e para todos os algoritmos da segunda parte do projeto.

Na terceira parte, o algoritmo foi exposto à uma base de dados muito maior e mais complexa, Olivetti Faces. Os resultados obtidos mostraram que para valores baixos de K, os índices de acerto são altos, maiores que 90%. Entretanto, conforme o valor de K aumenta, as métricas passam a apresentar valores muito mais baixos, de cerca de 50%.

Ainda poderiam ser feitas modificações significativas para que o tempo de processamento fosse menor, tal como paralelizar o código e implementar uma função de ordenação mais rápida. Mas, de forma geral, pode-se dizer que os objetivos foram satisfatoriamente atingidos, uma vez que o resultado das classificações foram idênticos aos de uma alternativa já muito bem consolidada, o Scikit Learn.

5 Referências

[1] N. S. Altman, “An introduction to kernel and nearest-neighbor nonparametric regression,” *The American Statistician*, vol. 46, no. 3, pp. 175–185, 1992.

[2] A. Smola and S. Vishwanathan, *Introduction to Machine Learning*. Cambridge University Press, 2008.

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

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

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