Software Impacts 15 (2023) 100459

Contents lists available at ScienceDirect

Software Impacts

journal homepage: www.journals.elsevier.com/software-impacts

Original software publication

PL-kNN: A Python-based implementation of a parameterless k-Nearest Neighbors classifier

Danilo Samuel Jodas [a,*], Leandro Aparecido Passos [b], Ahsan Adeel [b], João Paulo Papa [a]

[a] São Paulo State University, Bauru SP, Brazil
[b] School of Engineering and Informatics, University of Wolverhampton, UK

ARTICLE INFO

Keywords:
Machine learning
k-Nearest Neighbors
Classification
Clustering
Python

ABSTRACT

This paper presents an open-source implementation of PL-kNN, a parameterless version of the k-Nearest Neighbors algorithm. The proposed model, developed in Python 3.6, was designed to avoid the choice of the k parameter required by the standard k-Nearest Neighbors technique. Essentially, the model computes the number of nearest neighbors of a target sample using the data distribution of the training set. The source code provides functions resembling the Scikit-learn methods for fitting the model and predicting the classes of the new samples. The source code is available in the GitHub repository with instructions for installation and examples for usage.

Code metadata

Current code version	*1.0*
Permanent link to code/repository used for this code version	https://github.com/SoftwareImpacts/SIMPAC-2022-275
Permanent link to Reproducible Capsule	https://codeocean.com/capsule/6555988/tree/v1
Legal Code License	*MIT License.*
Code versioning system used	*git*
Software code languages, tools, and services used	*Python*
Compilation requirements, operating environments & dependencies	Python ≥ 3.6, numpy package
If available Link to developer documentation/manual	
Support email for questions	danilojodas@gmail.com

1. Introduction

The main challenges in machine learning tasks regard the proper selection of the prediction model and the further optimization of its hyperparameter values that properly fit the data distribution [1]. Regarding the latter aspect, one can notice the computational cost of seeking the hyperparameter values that improve the model's accuracy and prediction capacity. Depending on the model, several hyperparameters must be considered in the optimization procedure, thus leading to time-consuming efforts that may increase according to the dataset size [2,3]. For example, Random Forest, Support Vector Machines, and k-Nearest Neighbors (k-NN) [4] rely on some hyperparameter values for classification and regression tasks.

The k-Nearest Neighbors (k-NN) is a simple model that relies on the nearest neighbors concept. Firstly, it selects the k training samples close to the test instance. In classification tasks, the process continues by assigning the majority class determined by voting from the nearest neighbors of the target sample. In regression tasks, the target sample receives the average value computed from the nearest neighbors. Since it is deterministic in its conception, k-NN produces the same results

The code (and data) in this article has been certified as Reproducible by Code Ocean: (https://codeocean.com/). More information on the Reproducibility Badge Initiative is available at https://www.elsevier.com/physical-sciences-and-engineering/computer-science/journals.

* Corresponding author.
E-mail addresses: danilojodas@gmail.com (D.S. Jodas), l.passosjunior@wlv.ac.uk (L.A. Passos), ahsan.adeel@deepci.org (A. Adeel), joao.papa@unesp.br (J.P. Papa).

https://doi.org/10.1016/j.simpa.2022.100459
Received 29 November 2022; Received in revised form 14 December 2022; Accepted 14 December 2022

regardless of the number of iterations. The result's accuracy may differ depending on the number of neighbors specified by the hyperparameter k. Usually, a low number is selected empirically. Still, a fine-tuning procedure might be necessary to find the best value for k that sets a correct number of neighbors for specific data distribution, thus attaining more accurate results. Conversely, the computational cost tends to rise as the size of the data increases and becomes more complex.

To address the hyperparameter selection for the k-NN algorithm, we proposed a method to sidestep the setting of the k value in our previous study [5]. The proposed approach, called PL-kNN, computes the proper number of neighbors for predicting the test samples. Unlike the standard k-NN, which uses a fixed value for k, each sample's number of neighbors is different depending on the data distribution. Moreover, the implementation of the method is straightforward and adaptable to any programming language, although it was developed using Python 3.6 in this work.

2. Source-code design

Let $\mathcal{Y} = \{y_1, y_2, y_3, \ldots, y_n\}$ be the set of classes considered in the assessed dataset, where y_i represents the ith class. Also, let $\mathcal{T} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \ldots, \boldsymbol{x}_m\}$ be the set of samples of the dataset represented by a feature vector denoted as $x_j \in R^m$, being x_j the feature vector of the jth sample. Each sample x_j is assigned to a class y_i such that the pair (x_j, y_i) is used in the subsequent training and testing of the classifier. Formally speaking, this step involves partitioning the samples such that $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2$ and $\mathcal{T}_1 \cap \mathcal{T}_2 = \emptyset$, where $\mathcal{T}_1$ and $\mathcal{T}_2$ denote the training and testing sets, respectively. First, the proposed method divides the training data $\mathcal{T}_1$ into n clusters according to its number of classes in $\mathcal{Y}_1$, where $\mathcal{Y}_1$ is the set of labels of the training instances in $\mathcal{T}_1$. Afterward, we seek the cluster's center close to the test data $t \in \mathcal{T}_2$. Then, the model selects the nearest neighbors using a semicircle computed as the distance between the test sample and the closest cluster's center. Finally, we determine the class s of t as the class with the higher frequency among all nearest neighbor samples. This approach is effective when instances of different classes are not properly separated in the data distribution (see the examples illustrated in Fig. 1).

The proposed library was developed in Python 3.6 using only the NumPy package for numerical operations. The setup is simple and compatible with the newest release of the Python programming language. The main file is **pl_nn/pl_nn.py** in the GitHub repository, which implements the class of the PL-kNN model. Essentially, the most important functions of the class are described as follows:

- **fit**: it performs the training stage of the model. In summary, it receives the training samples and the respective labels in order to determine the centers of the classes;
- **predict**: it receives a list with the test samples to be predicted. Essentially, it seeks the nearest neighbors using a semicircle computed as the distance between the test sample under analysis and its nearest class center determined in the training phase of the model. Afterward, the test sample receives the class computed from a weighted majority voting from those nearest neighbor's instances (see Jodas et al. [5] for more details).

In addition to the methods, the class also contains the following attributes:

- **X_train**: it holds the samples of the training set;
- **classes**: it stands for the label of each class;
- **centers**: it represents the samples that stand for the centers of each class distribution.

Fig. 1 illustrates how PL-kNN selects the neighbors of a test sample considering a synthetic dataset composed of 100 samples and 2 features. In both cases, it is possible to notice the correct neighbors selected for each test sample regarding the complexity of the data distribution and the compatibility of the classes of the neighbor's samples with the class assigned to each test instance.

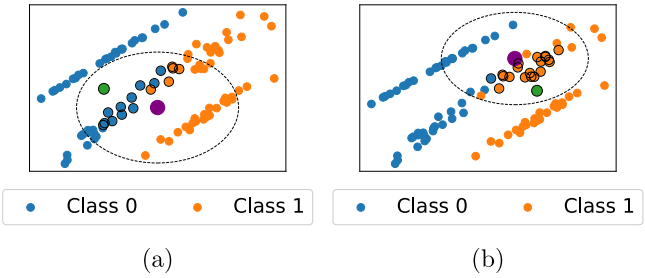

Fig. 1. Illustration of the nearest neighbors selected for the test sample depicted as the purple dot: (a) A test sample belonging to class 0; (b) A test sample belonging to class 1. The selected nearest neighbors are the dots with black contours inside the circle computed as the distance between the test sample and the center of the data distribution. The green dot stands for the data distribution center. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

3. Software impacts

The proposed implementation aims to promote additional research in designing classifiers that circumvent the necessity for hyperparameter definition, like the Optimum-Path Forest classifier proposed by Papa et al. [6]. We selected k-NN due to its deterministic behavior and simplicity for implementation. In the previous study [5], we reported several improvements attained by the PL-kNN model compared to the baseline k-NN algorithm optimized using a hyperparameter optimization procedure. The proposed model achieved the best results in seven out of eleven datasets and statistical similarity in two of them compared to the best-performing algorithm. Using a simple approach, PL-kNN is easily adaptable to other programming languages and customizable to include a set of functionalities that allows the exploitation of the prediction behavior. Such an approach is essential to assess the model results and the data distribution complexity concerning the context under consideration. Furthermore, the Python implementation is designed to provide convenience and integration with other frameworks. The proposed design is deployable into some objects of the Scikit-learn package, like a sequence of processes assembled by the Scikit-learn pipeline class. It may be helpful while testing several classifiers and different settings of parameters.

4. Conclusions and future works

The proposed library is an easy-to-use implementation that resembles the Scikit-learn package's functions concerning the model's training and calling of new predictions. Due to its simplicity, researchers and developers may exploit its functionalities and use or extend the code for their own investigation.

At the present stage of this work, the provided implementation supports the object's instantiation and calling of its function to train and test the model's performance. Moreover, the source code is only designed for classification purposes; therefore, we intend to extend the source code implementation for regression tasks in future works. Improvements in the neighbors' selection are also expected to identify the cases where more neighboring samples are required to increase the prediction accuracy.

CRediT authorship contribution statement

Danilo Samuel Jodas: Methodology, Software, Writing – original draft. **Leandro Aparecido Passos:** Methodology, Writing – original draft. **Ahsan Adeel:** Writing – review & editing. **João Paulo Papa:** Funding acquisition, Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors are grateful to FAPESP, Brazil grants #2013/07375-0, #2014/12236-1, #2017/02286-0, #2018/21934-5, #2019/07665-4, and #2019/18287-0, Engineering and Physical Sciences Research Council (EPSRC), United Kingdom grant EP/T021063/1, CNPq, Brazil grants #307066/2017-7, and #427968/2018-6, and Petrobras, Brazil grant #2017/00285-6.

References

[1] Li Yang, Abdallah Shami, On hyperparameter optimization of machine learning algorithms: Theory and practice, Neurocomputing 415 (2020) 295–316.
[2] Xianping Du, Hongyi Xu, Feng Zhu, Understanding the effect of hyperparameter optimization on machine learning models for structure design problems, Comput. Aided Des. 135 (2021) 103013.
[3] Jia Wu, SenPeng Chen, XiYuan Liu, Efficient hyperparameter optimization through model-based reinforcement learning, Neurocomputing 409 (2020) 381–393.
[4] Pádraig Cunningham, Sarah Jane Delany, K-nearest neighbour classifiers - a tutorial, ACM Comput. Surv. 54 (6) (2021).
[5] Danilo Samuel Jodas, Leandro Aparecido Passos, Ahsan Adeel, João Paulo Papa, PL-kNN: A Parameterless Nearest Neighbors Classifier, in: 2022 29th International Conference on Systems, Signals and Image Processing, Vol. CFP2255E-ART, IWSSIP, 2022, pp. 1–4.
[6] J.P. Papa, A.X. Falcão, C.T.N. Suzuki, Supervised pattern classification based on optimum-path forest, Int. J. Imaging Syst. Technol. 19 (2) (2009) 120–131.