Random k-nearest-neighbour imputation (C[28])

Yuchao Wu

{wuyucha}@uwindsor.ca

I. Introduction

The Random K-nearest-neighbor (RKNN) is an algorithm that can be used to impute randomly distributed missing values of the input dataset [1].

II. Project Overview

In this project, the algorithm of Random KNN imputation is implemented to process incomplete dataset with various dimensions from 1000 by 26 to 100000 by 51. The computed NRMS is reasonable whereas the time spent on each incomplete dataset varies from a blink to a few hours with the increasement of data dimensions.

A. *The property of KNN*

As shown in Figure 1, the KNN is used for the imputation of missing values of the input dataset, that is, to figure out the closest *k* samples within the same feature of the missing value, take the mean of the data nearby (based on distance metric, such as Euclidean distance) to fill in the missing value [3].

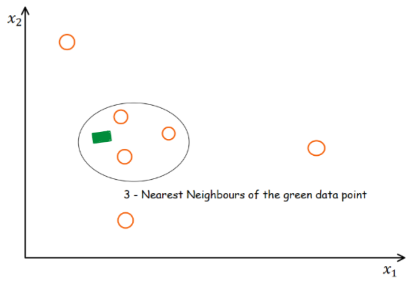

Fig. 1. The mechanism of KNN [4].

The value k of the nearest neighbor is commonly selected by the function $k = \sqrt{m}$ [1].

Eq. (1) expresses the computation of Euclidean distance between two tuples X_1 and X_2 [3].

$$\text{dist}\,(\boldsymbol{X_1}, \boldsymbol{X}_2) = \sqrt{\sum_{i=1}^{m} (x_{1i} - x_{2i})^2} \tag{1}$$

Where $X_1 = (x_{11}, x_{12}, \dots, x_{1n}), X_2 = (x_{21}, x_{22}, \dots, x_{2n})$

B. *Data nomalization*

Sometimes the attribute contains a larger range compared to other attributes, which will influence the computation of Euclidean distance; thus, we can use the min-max normalization method to transform the numeric attribute of the input dataset to the range from 0 to 1. Eq. 2 shows the transformation process [3].

$$v' = \frac{v - \underset{A}{min}}{\underset{A}{max} - \underset{A}{min}} \tag{2}$$

Where v' is the transformed value from its original value v of the attribute A of a dataset, $\underset{A}{max}$ is the maximum value of attribute A, and $\underset{A}{min}$ is the minimum value of attribute A.

III. Random KNN Imputation

A. *The property of RKNN*

Let $\{Z_i\}, i = 0, \dots, m$ be a set of i.i.d. random variables, which has a dimension of n. When $i \neq i'$, Z_i is independent of $Z_{i'}$. Also, let l be the number of missing values of Z_0, where $l < n$. Let $\mathbf{Z} = \binom{\mathbf{Y}}{\mathbf{X}}$, where $\mathbf{Y} = [Y_1 \dots Y_n], \mathbf{X} = [X_1 \dots X_n]$, and let $d(X, X')$ represents the Euclidean distance between X and X'. [1]

B. *Pseudom code RKNN Imputation*

TABLE I. RKNN Algorithm [1]

Input: an input dataset with m+1 rows and n columns.

m – the number of samples in each attribute of the training dataset.

n – the number of attributes

k – the number of nearest neighbors

Output: the imputed value Y_l of the missing data

Algorithm:

```
1. Let $k = \sqrt{m}$ ; % Set number of nearest neighbor, $k$ should be rounded upward.
2. Delete the row which has n missing values.
3. Figure out the indexes of missing value.
4. Figure out the total number of missing values (noted as $L$).
5. for iteration = 1 : $L$
6. for i = 1 : m
7.   % Compute Euclidean Distance,
     $d_i = d(X_0, X_i) = \sqrt{\sum_{i=1}^{m} (x_0 - x_i)^2}$ +M$\chi$ (if $Y_i$ has missing value)
```

```
    Where, $\chi$ is an indicator which returns 1 if $Y_i$ has missing value,
    otherwise, return 0. M is a penalty if $Y_i$ has missing value; $X_0$ is given.
8.  for iterate = 1:$k$
9.  Sort indices of Euclidian distances by ascending order, select first to kth
    smallest values, noted as idx_KNNdata,
10. end
11. Find k nearest neighbors, noted as KNNdata
12. end
13. Identify the indexes (noted as r) of the KNNdata, where $r \subset \{1, \ldots, m\}$,
    $|r| = k$.
14. I is randomly selected from r.
15. Find one of the $k$ nearest samples around $Y_0$ based on the index I;
16. The $I^{th}$ element (noted as $Y_I$) will be used to impute the jth missing value.
17. end
```

IV. Time Complexity Analysis

A. Impute all incomple datasets

Consider *i* complete datasets and *i* folders with *j* incomplete datasets each. Firstly, *sort_nat* function sorts the names of complete dataset by natural order, *O(i)*, and then sorts the name of folders of incomplete datasets, *O(i)*. Then, a for loop is iterated *i* times, *O(i)*, in which *sort_nat* function sorts names of incomplete datasets by natural order, *O(j)*, then, another for loop is iterated *j* times, *O(j)*, in which, perform Random KNN imputation based on *k* nearest neighbors. In RKNNI process, first for loop is repeated *L* times, *O(L)*, inside this loop, *pdist2* function only computes *m* pairs of Euclidean distance, also finds the indices of *k* smallest distances, *O(mk)*. Thus, the overall time complexity of RKNNI is as follows:

$$O(i+i+i(j+j(L(mk))) = O(i+i+ij+ijLmk)$$

As $i < j$, $i*j < ijLmk$, therefore, the time complexity is:

$$O(ijLmk)$$

B. Impute single incomplete dataset

Only consider imputing single incomplete dataset. Which means $O(ij) = 1$, the overall time complexity is $O(Lmk)$.

V. Implimentation

The Random KNN Imputation is implemented using MATLAB 2020a, requires Statistics and Machine Learning Toolbox. Ran in Microsoft Windows with the AMD Ryzen 5 4600H CPU at 3.0 GHz and 16GB RAM.

The input data includes incomplete dataset stored in *csvdata*, and complete dataset stored in *origiData.*

The output is imputed dataset *ipt*, *NRMS*, and *tEnd*, where *NRMS* is a 44 by 14 matrix which represents NRMS of 616 incomplete datasets, *tEnd* represents runtime spent on the imputation process of each of 616 incomplete datasets.

VI. Experimental Results

A. Experimental Setting

Figure 1 is a demo to show that in experiment 1, the incomplete dataset is split into imputing set and trainset. In which red means missing values, green means given values in each of the columns where missing values located; yellow means complete dataset for computing Euclidian distance. This experimented is represented by MATLAB function *RandomKNNImputeV4.m* in project folder, and the overall imputing process is performed by main program *C28_RKNN_Impute_V1.m*

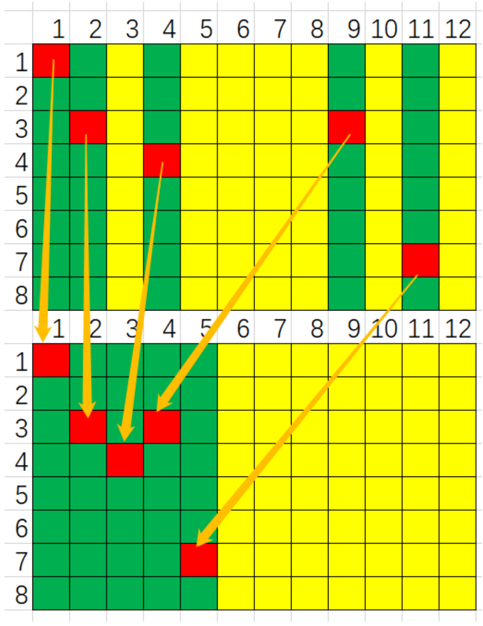

Fig. 2. Experiment 1 demo: Split incomplete dataset into two sets (Left set for imputing and right set for training/Euclidian distance computing).

Figure 2 is another demo to show that in experiment 2, the incomplete dataset is imputed without splitting into two sets. In which red means missing values, green means given values in each of the columns where missing values located.

Now assume that we're going to impute first missing value in 1st row and 1st column. Where blue means given values in the same row of fist missing value; yellow means given values used for computing weighted Euclidian distance, where the weight is defined as [4]:

$$d_{xy} = \sqrt{weight \; * squared\ distance\ from\ present\ coordinates} \quad (2)$$

$$weight = \frac{Total\ number\ of\ coordinates}{Number\ of\ present\ coordinates} \quad (3)$$

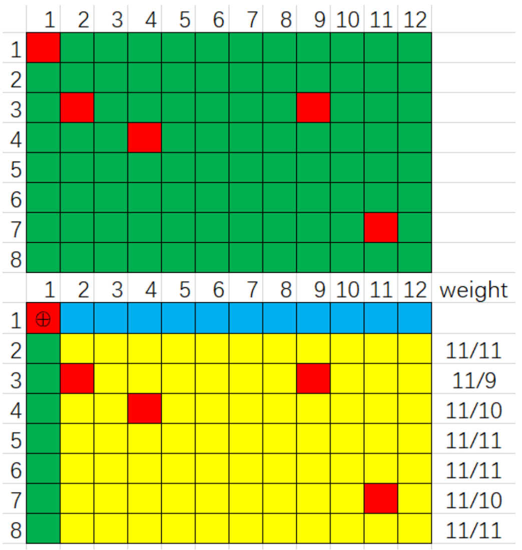

Fig. 3. Experiment 2 demo: using rows with missing values to compute weighted Euclidian distance

This experiment 2 is represented by MATLAB function *RandomKNNImputeV5.m* in project folder, and the overall imputing process is performed by main program *C28_RKNN_Impute_V2.m*

As there are 14 complete datasets, and 14 folders with 44 incomplete datasets each. The main program uses two for loops to import complete dataset and incomplete datasets automatically. The first for loop will iterate 14 times and the second for loop will iterate 44 times.

For instance, the main program will import *Data_1.csv* from the folder *Complete datasets*, and 1st incomplete dataset *Data_1_AE_1%.csv* in the folder *Incomplete datasets\Data 1*, after the missing values are imputed, Runtime is recorded, and NRMS is computed, then the program will import 2nd incomplete dataset in the same folder. After all incomplete datasets are process, the program will write NRMS and Runtime data to *Table_NRMS_Write.xlsx.*

As of the parameter *k* for nearest neighbor, which is defined in Table 1. That is, *k* will be decided by the dimension of incomplete dataset stored in *csvdata.*

B. *Results Analysis based on Experiment 1*

By observing Figure 4, it can be shown that missing type *AG* in data 7 consumed more than twice the time as consume by other types in the dataset. Data 10 shows similar phenomenon, where the missing type C take far more time than any other types.

Figure 5 shows that the average NRMS in first 5 missing types is lower than that in last 6 missing types from data 1 to 8 and data 10 to 13.

Figure 6 shows that from data 1 to 10 the average runtime increase slowly and all average computing time are less than 100 seconds; however, from data 11 to 14, the average runtime increased sharply from around 50 seconds to around 2600 seconds; furthermore, higher ratio of missing data shows longer computing time compared with that of lower ratio of missing data. In data 14, the average runtime of dataset with 20% missing ratio doubled that of dataset with10% missing ratio , and tripled that of dataset with 5% missing ratio, and is tenfold of that of dataset with 1% missing ratio.

Figure 7 shows that dataset with higher ratio of missing data has higher NRMS than that of dataset with lower ratio of missing data. In each of 14 data, the average NRMS in dataset with 20% missing ratio almost doubled that of dataset with 10% missing ratio, this means that the NRMS increases with the ratio of missing data grow higher.

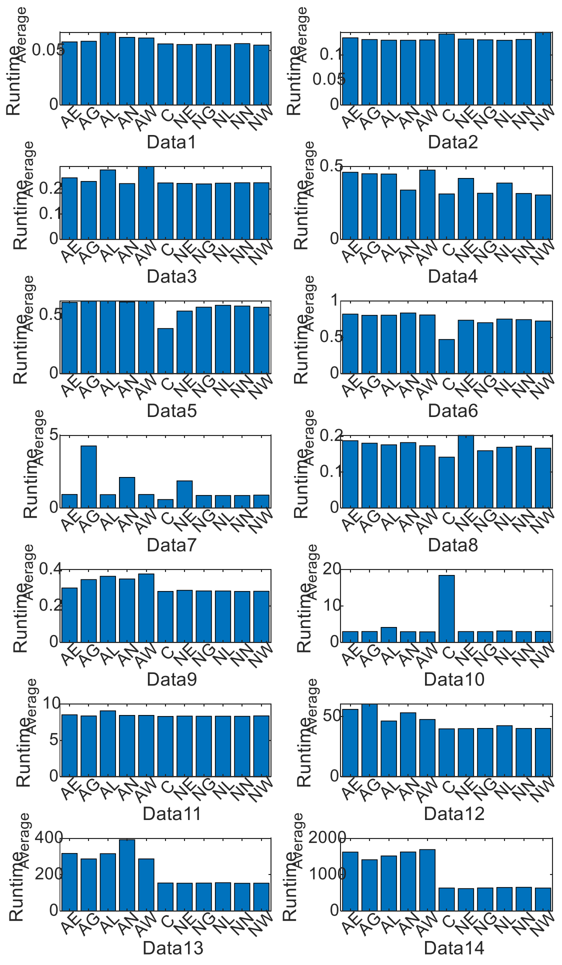

Fig. 4. Average runtime vs. different missing types

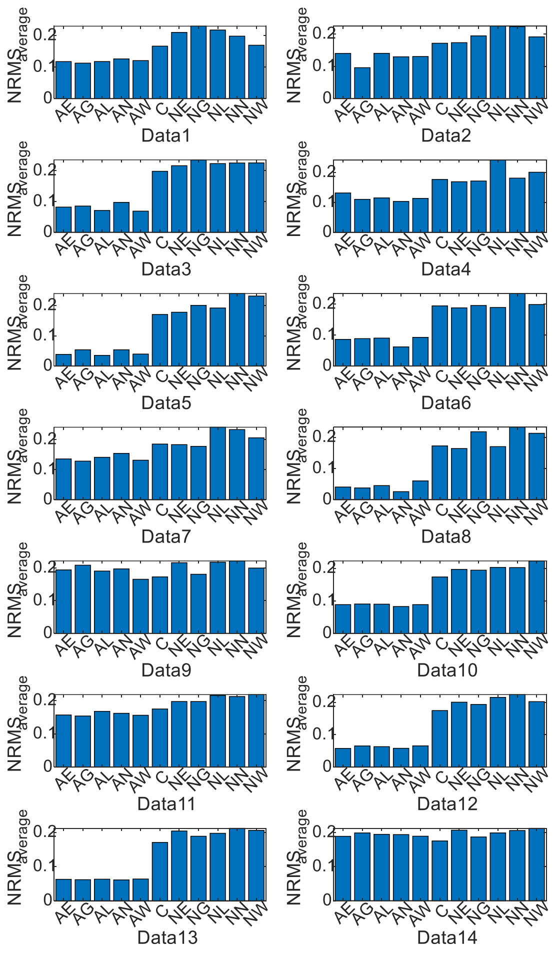

Fig. 5. Average NRMS vs. different missing types

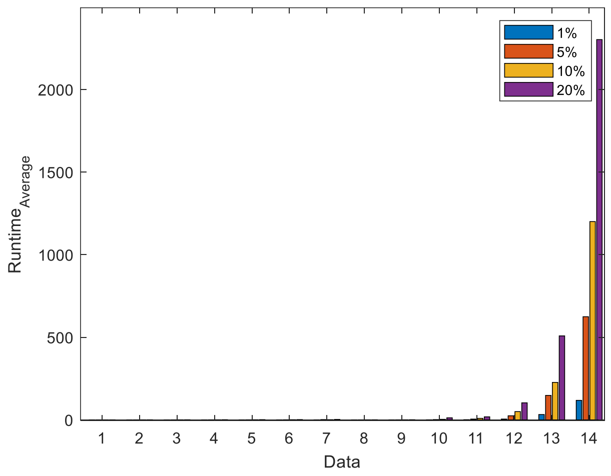

Fig. 6. Average runtime vs. different ratio of missing data

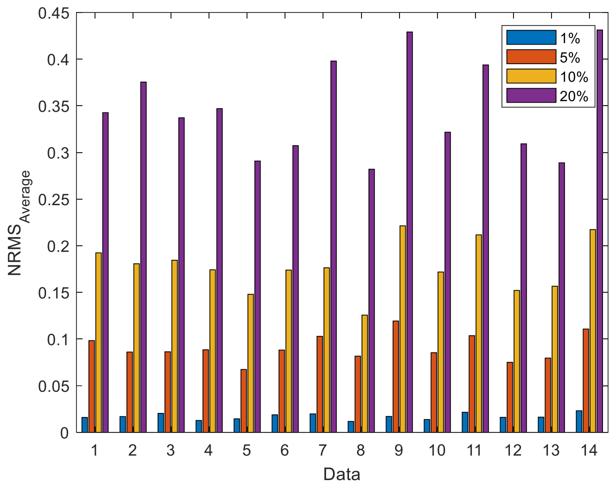

Fig. 7. Average NRMS vs. different ratio of missing data

References

[1] "Convergence of random k-nearest-neighbour imputation Fredrik A. Dahl," *Computational Statistics & Data Analysis*, vol. 51, pp. 5913–5917, 2007.

[2] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*. Oxford, England: Morgan Kaufmann, 2012.

[3] K. Chomboon, P. Chujai, P. Teerarassammee, K. Kerdprasop, and N. Kerdprasop, "An empirical study of distance metrics for k-nearest neighbor algorithm," in *The Proceedings of the 2nd International Conference on Industrial Application Engineering 2015*, 2015.

[4] K. R. Chowdhury, "KNNImputer: A robust way to impute missing values (using Scikit-Learn)," *Analytics Vidhya*, 13-Jul-2020. [Online]. Available: https://www.analyticsvidhya.com/blog/2020/07/knnimputer-a-robust-way-to-impute-missing-values-using-scikit-learn/. [Accessed: 02-Jul-2022].