FEP FACULDADE DE ECONOMIA
UNIVERSIDADE DO PORTO

2018/2019

Master of Data Analysis and Decision Support Systems

Data Bases and Programming

KNN classifier and regressor class in R with hyperparameter tuning and preprocessing embedded

Professors: Professor Rui Leite

Students: Jorge Ferreira, up201204029

Pedro Esmeriz, up201202790

Tiago Craveiro, up200908673

This project concerns a K-Nearest Neighbors model built as a class in R where several procedures can be employed “off the shelve”, without requiring any further work from the end user other than providing a training dataset and the dataset upon which he desires to predict the outcome.
Therefore, and in summary, to enable this degree of simplicity, the following methods have been implemented within this class and are used either to enable better results of the model and/or to improve the quality of the training set. Some methods have also some functions of their own that will also be explain within each segment.

1. Initiate the model

In here the user defines how the object will process the data that will later be fed, by defining the distance function to determine the neighbors, the number of neighbors to use, the tie break criteria that should be used and the NA handling of data desired.

2. **Fit**

After creating the model, the user feeds the training datasets, after which several functions will take place to process the data into a dataset viable of being used to predict the class of the test datasets. The following subchapters include the functions that can be activated within this segment.

2.1Dummy_vars_encoding

This is one of the first steps on the class where categorical attributes are transformed into a numeric one through a binarization process: previous column *X = {x1, x2, …, xn}*, where xi concerns each distinct value, is transformed to columns *dummy_x1, dummy_x2, … dummy_xn*. These columns will hold either 1 or 0, depending if the previous value is respective to the one the posterior column wants to represent (example: if *x1* then *dummy_x1* will be 1 for the same position in the matrix). With support to this function, the user can submit any kind of data that it will be rectified to enable the nearest neighbor search.

2.2NA handling

In cases where the datasets are submitted with NA rows, the user can select between (i) removing the NA from the analysis and keeping only the columns where full information, row-wise, is provided and (ii) filling the NA based on the remaining data, through method knn_na.

2.2.1 KNN_NA

This algorithm starts by dividing the dataset into two: one with the rows where at least 1 NA is present per row, hereby called *filling_df,* and another where every row possesses the full data, hereby called *base_data*. The procedure is performed row by row on the *filling_df*, where in each row, for each NA, a K-NN search is performed to determine the value of the NA by searching on the *base_data* the nearest neighbors. After each row is fully filled, it is then added to the *base_data* the process continues until all rows have been processed.

3. Predict

In this segment the user submits the test dataset, that will also be subject to the dummy_vars_encoding function and the results will be the final predictions for each row. In order to produce it, the test dataset go through two functions.

3.1 KNN Search

In this function, the distance of the test set and the training set is computed, where the user can define (i) either the euclidean or the manhattan distance and (ii) the number of neighbors k desired for each observation.

3.2 KNN Output Selector

Finally the k neighbors are analysed and the final prediction of the class, in case of classification, or value, in case of regression, is presented. For regression, the output concerns the mean of all neighbors. Concerning classification, it depends on the tie-break criteria where the user can choose between (i) random selection among top frequent class and (ii) reducing k in a recursive manner until an undisputed class emerges as the most frequent.

Class model hyperparameters and its default values: knn(*k = 3, distance = "euclidean", tie_break="reduce_k", na_handling="remove NA")*

Parameter: ***k: int, optional (default =3)***

Number of neighbors to use.

metric : string (default = 'euclidean')

*Distance metric. Valid values for metric are: ['euclidean', '*manhattan']

tie_break: string (default ='reduce_k')

*Tie break criteria on class decision. Valid values for metric are: ['reduce_k', '*random'']

na_handling: string (default='*remove NA')*

How to deal with missing values. alid values for metric are: ['remove NA', 'knn NA']

Code example calling class. It is possible to see the abstraction:

```
#import data
df = read.csv('./Documents/MADSAD/Programming and Databases/dataset3.csv')
X_train= df[ , (names(df)  != 'class')]
X_train
y_train = df[ , (names(df)  = 'class')]
y_train
x_test =  data.frame(age=c(10,15,16), Height=c(1.2,1.6,1.7), state=c('B','A','C'))


# create some knn
k_nn <- new("knn", k=3, distance="euclidean", tie_break='reduce_k', na_handling='knn_na')

#get model type
model_type(k_nn)

#train some knn
k_nn<-fit(k_nn, X_train=X_train, y_train = y_train)

#get model type
model_type(k_nn)

#get knn predictions
y_test<-predict(k_nn, x_test)
```