Programming Nearest Neighbor in C++

Advanced programming

Ignacio Almodóvar & Andrés Mejía

2021

# Translate R code into C++

In order to translate the code given in the task, we just needed to change the syntaxis from R to C++. These two languages at a basic level are not that different. Therefore, the changes that we had to made were related to declaring variables within their types and writing “;” at the end of every row of code. Also, there were a few changes on the way that loops are writen.

// [[Rcpp::export]]  
int my\_knn\_cpp(NumericMatrix X, NumericVector X0, NumericVector y) {  
   
 int nrows= X.nrow();  
 int ncols= X.ncol();  
   
 double closest\_distance = 99999999;  
 int closest\_output=-1;  
 int closest\_neighbor=-1;  
  
 double diff=0;  
   
 for(int i=0; i<nrows;i++){  
 double distance =0;  
 for(int j=0; j<ncols; j++){  
 diff=X(i,j)-X0[j];  
 distance+=diff\*diff;  
   
 }  
   
 if(distance<closest\_distance){  
 closest\_distance=distance;  
 closest\_output=y[i];  
 closest\_neighbor=i;  
 }  
   
 }  
   
 return closest\_output;  
}

## Compile the C++ code with sourceCpp. Show in your report that you have used sourceCpp.

The function mentioned above was written in a .cpp file. In order to be able to use it in a R script we need to add this file using a function from the library Rcpp called “sourceCpp()”.

library(Rcpp)  
library(FNN)  
library(microbenchmark)  
library(dplyr)

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

sourceCpp('src/function\_c.cpp')

The data that we are going to run all this functions is going to be the same as the one used in the example given.

data("iris")  
X <- iris[,-5]  
y <- iris[,5]  
X <- as.matrix(X)  
y <- as.integer(y)  
  
# This is the point we want to predict  
X0 <- c(5.80, 3.00, 4.35, 1.30)

Now we can run our function coded in C++.

my\_knn\_cpp(X,X0,y)

## [1] 2

## Check that you get the same results than the knn that belongs to library FNN or classif (choose one)

Using the function given in the task and the one from the library KNN, we can compute the distances for our sample and check that we get the same results in our c++ function.

my\_knn\_R(X,X0,y)

## [1] 2

d=FNN::knn(X, matrix(X0, nrow = 1), y, k=1)  
d[[1]]

## [1] 2  
## Levels: 2

## Use the library microbenchmark in order to determine whether the C++ version is faster than:

Using “microbenchmark()” function, we can calculate the time taken for each function to be computed.

### The R version of the code

microbenchmark(my\_knn\_cpp(X,X0,y),my\_knn\_R(X,X0,y))

## Unit: microseconds  
## expr min lq mean median uq max  
## my\_knn\_cpp(X, X0, y) 3.354 4.396 112.946 7.4775 14.4895 7687.245  
## my\_knn\_R(X, X0, y) 925.363 1050.896 1676.328 1361.4095 1668.3945 7867.000  
## neval cld  
## 100 a   
## 100 b

As expected, the C++ function is 287.50 times faster than the R one. This is due to the fact that R is an interpreted language whereas C++ is a compiled language.

### The knn of FNN / classif

microbenchmark(my\_knn\_cpp(X,X0,y),FNN::knn(X, matrix(X0, nrow = 1), y, k=1))

## Unit: microseconds  
## expr min lq mean  
## my\_knn\_cpp(X, X0, y) 3.569 6.2865 16.31105  
## FNN::knn(X, matrix(X0, nrow = 1), y, k = 1) 464.925 504.3325 924.97615  
## median uq max neval cld  
## 8.9250 17.3235 111.243 100 a   
## 777.4545 1175.9585 3072.324 100 b

In this case we have obtained again that the function in C++ is 131.8396 times faster. Checking the “FNN::knn()” function (using the f2 key) we can see that it is better optimized than the function “my\_knn\_R()”. However, it still runs in pure R. Therefore it gets better but it is not going to be as faster as C++.

# Create a function that calculates Minkowsky distance.

We have created a function called “Mink\_dist” inside the package “knnH”. This function takes two vectors and an argument that, if positive, then it is exponent p, and if zero or negative, then it means .

double Mink\_dist(NumericVector x,NumericVector y,double p) {  
   
 double cumulant=0;  
 double expre =0;  
   
 if(p<=0){  
 for(int i=0; i<x.length();i++){  
 expre=fabs(x[i]-y[i]);  
 if(expre>cumulant){  
 cumulant=expre;  
 }  
 }  
 return cumulant;  
 }  
   
 for(int i=0; i<x.length();i++){  
 expre=pow(fabs(x[i]-y[i]),p);  
 cumulant+=expre;  
 }  
   
 double k=1/p;  
 cumulant=pow(cumulant,k);  
   
 return cumulant;  
}

Here we show some examples:

#install.packages("/Users/ignacioalmodovarcardenas/Desktop/Advanced programming/knnH\_0.9.tar.gz",  
# repos=NULL,type = "source")  
library(knnH)  
  
a=1:3  
b=4:6  
  
Mink\_dist(a,b,2)

## [1] 5.196152

sum(abs(a-b)^2)^(1/2)

## [1] 5.196152

Mink\_dist(a,b,7)

## [1] 3.509792

sum(abs(a-b)^7)^(1/7)

## [1] 3.509792

Mink\_dist(a,b,-1)

## [1] 3

min(abs(a-b))

## [1] 3

# Standarization and Normalization

We created the following functions included the package “knnH” to normalize and standardize our data.

* stand\_data(): Standardize a vector.
* stand\_data\_Matrix(): Standardize each column of a matrix.
* minmax\_data(): Normalize a vector
* minmax\_data\_Matrix(): Normalize each column of a vector.

// [[Rcpp::export]]  
NumericVector minmax\_data(NumericVector x) {  
   
 double minx=min(x);  
 double maxx=max(x);  
   
 NumericVector xs=x-minx;  
 xs=xs/(maxx-minx);  
   
 return xs;  
}  
  
  
// [[Rcpp::export]]  
NumericVector stand\_data(NumericVector x) {  
   
 double meanx=mean(x);  
 double sdx=sd(x);  
 NumericVector xs=x-meanx;  
 xs=xs/sdx;  
   
 return xs;  
}  
  
// [[Rcpp::export]]  
NumericMatrix stand\_data\_Matrix(NumericMatrix X) {  
 NumericMatrix X\_outmm=clone(X);  
 for(int i=0;i<X\_outmm.ncol();i++) {  
 X\_outmm(\_,i) = stand\_data(X\_outmm(\_,i));  
 }  
 return(X\_outmm);  
}  
  
// [[Rcpp::export]]  
NumericMatrix minmax\_data\_Matrix(NumericMatrix X) {  
 NumericMatrix X\_outm=clone(X);  
 for(int i=0;i<X\_outm.ncol();i++){  
 X\_outm(\_,i)=minmax\_data(X\_outm(\_,i));  
 }  
 return X\_outm;  
}

Some examples:

A=stand\_data\_Matrix(X)  
for(i in 1:ncol(A)){  
 paste0("La media de la columna ",i, " es ",mean(A[,i]), " y la desviacion ",sd(A[,i])) %>% print()  
}

## [1] "La media de la columna 1 es -4.46260189175193e-16 y la desviacion 1"  
## [1] "La media de la columna 2 es 2.17851814423668e-16 y la desviacion 1"  
## [1] "La media de la columna 3 es -1.11195774810113e-17 y la desviacion 1"  
## [1] "La media de la columna 4 es -3.66304931988711e-17 y la desviacion 1"

A=minmax\_data\_Matrix(X)  
for(i in 1:ncol(A)){  
 paste0("El mínimo de la columna ",i, " es ",min(A[,i]), " y el maximo ",max(A[,i])) %>% print()  
}

## [1] "El mínimo de la columna 1 es 0 y el maximo 1"  
## [1] "El mínimo de la columna 2 es 0 y el maximo 1"  
## [1] "El mínimo de la columna 3 es 0 y el maximo 1"  
## [1] "El mínimo de la columna 4 es 0 y el maximo 1"

# Final function

Using all these functions we have built another function that uses all these methods. This main function is also included in the package that we have built “knnH”.

// [[Rcpp::export]]  
int knn\_completa(NumericMatrix X, NumericVector X0, NumericVector y, int modo, int mk) {  
   
 NumericMatrix X\_out=clone(X);  
 NumericVector X0\_out=clone(X0);  
   
 if(modo==0 || modo==1){  
 if(modo==0){  
 X\_out=stand\_data\_Matrix(X\_out);  
 for(int i=0;i<X.ncol();i++){  
 double meanx=mean(X(\_,i));  
 double sdx=sd(X(\_,i));  
 X0\_out[i]=(X0[i]-meanx)/sdx;  
 }  
   
 }  
 else{  
   
 X\_out= minmax\_data\_Matrix(X\_out);  
 for(int i=0;i<X.ncol();i++) {  
 double minx=min(X(\_,i));  
 double maxx=max(X(\_,i));  
 X0\_out[i]=(X0[i]-minx)/(maxx-minx);  
 }  
 }  
 }  
   
 else{  
 printf("Fourth argument must be 0 or 1");  
 return(NULL);  
 }  
   
 int nrows= X\_out.nrow();  
   
 double closest\_distance = 99999999;  
 int closest\_output=-1;  
 int closest\_neighbor=-1;  
   
 double distance=0;  
   
 for(int i=0; i<nrows;i++){  
 distance=Mink\_dist(X\_out(i,\_),X0\_out,mk);  
   
 if(distance<closest\_distance){  
 closest\_distance=distance;  
 closest\_output=y[i];  
 closest\_neighbor=i;  
 }  
   
 }  
   
 return closest\_output;  
}

Now we test the function:

knn\_completa(X,X0,y,modo = 1,mk = 2)

## [1] 2