Assignmment 1

Daniel Alonso

November 20th, 2020

Importing libraries

```
library(Rcpp)
library(microbenchmark)
```

Teacher example (this code is NOT mine)

```
my_knn_R = function(X, X0, y){
  # X data matrix with input attributes
  # y response variable values of instances in X
  # X0 vector of input attributes for prediction

  nrows = nrow(X)
  ncols = ncol(X)

  # One of the instances is going to be the closest one:
  #   closest_distance: it is the distance , min_output
  closest_distance = 99999999
  closest_output = -1
  closest_neighbor = -1

  for(i in 1:nrows){

    distance = 0
    for(j in 1:ncols){
      difference = X[i,j]-X0[j]
      distance = distance + difference * difference
    }

    distance = sqrt(distance)

    if(distance < closest_distance){
      closest_distance = distance
      closest_output = y[i]
      closest_neighbor = i
    }
  }
  closest_output
}
```

Testing teacher example (This code is NOT mine)

```
# X contains the inputs as a matrix of real numbers
data("iris")
# X contains the input attributes (excluding the class)
X <- iris[,-5]
# y contains the response variable (named medv, a numeric value)
y <- iris[,5]
# From dataframe to matrix
X <- as.matrix(X)
# From factor to integer
y <- as.integer(y)
# This is the point we want to predict
X0 <- c(5.80, 3.00, 4.35, 1.30)
# Using my_knn and FNN:knn to predict point X0
# Using the same number of neighbors, it should be similar (k=1)
my_knn_R(X, X0, y)
```

```
## [1] 2
```

```
library(FNN)
FNN::knn(X, matrix(X0, nrow = 1), y, k=1)
```

```
## [1] 2
## attr(,"nn.index")
##      [,1]
## [1,]   96
## attr(,"nn.dist")
##           [,1]
## [1,] 0.2061553
## Levels: 2
```

Translating the teacher code into C++ into an Rccp function

```
cppFunction('
int knn_1(NumericMatrix X, NumericVector X0, NumericVector y) {
    int nrows = X.nrow();
    int ncols = X.ncol();

    double closest_distance = 99999999;
    double closest_output = -1;
    double closest_neighbor = -1;
    double difference = 0;

    int i;
    int j;

    for (i = 0; i < nrows; i++) {

        double distance = 0;
        for (j = 0; j < ncols; j++) {
            difference = X(i,j) - X0(j);
            distance = distance + difference * difference;
        }

        distance = sqrt(distance);

        if (distance < closest_distance) {
            closest_distance = distance;
            closest_output = y(i);
            closest_neighbor = i;
        }
    }
    return closest_output;
}')
```

Testing Rcpp translation

```
knn_1(X, X0, y)
```

```
## [1] 2
```

```
library(FNN)
FNN::knn(X, matrix(X0, nrow = 1), y, k=1)
```

```
## [1] 2
## attr(,"nn.index")
##      [,1]
## [1,]   96
## attr(,"nn.dist")
##           [,1]
## [1,] 0.2061553
## Levels: 2
```

Benchmarking differences in runtime between R version and Rcpp version

R version

```
microbenchmark(my_knn_R(X, X0, y))
```

```
## Unit: microseconds
##                expr     min       lq     mean  median      uq      max neval
##  my_knn_R(X, X0, y) 706.651 730.9565 773.7613 750.292 771.472 2054.493   100
```

We can see that our mean runtime for the R version is more than 800 microseconds

FNN version

```
microbenchmark(FNN::knn(X, matrix(X0, nrow = 1), y, k=1))
```

```
## Unit: microseconds
##                                         expr     min      lq     mean  median
##  FNN::knn(X, matrix(X0, nrow = 1), y, k = 1) 223.661 226.866 236.5668 228.696
##        uq     max neval
##  236.5415 455.291   100
```

We can see that our mean runtime for the Rcpp version is of under 250 microseconds

Rcpp version

```
microbenchmark(knn_1(X, X0, y))
```

```
## Unit: microseconds
##             expr   min    lq     mean median    uq     max neval
##  knn_1(X, X0, y) 4.311 4.446 13.83541  4.521 4.601 933.852   100
```

We can see that our mean runtime for the Rcpp version is of under 14 microseconds