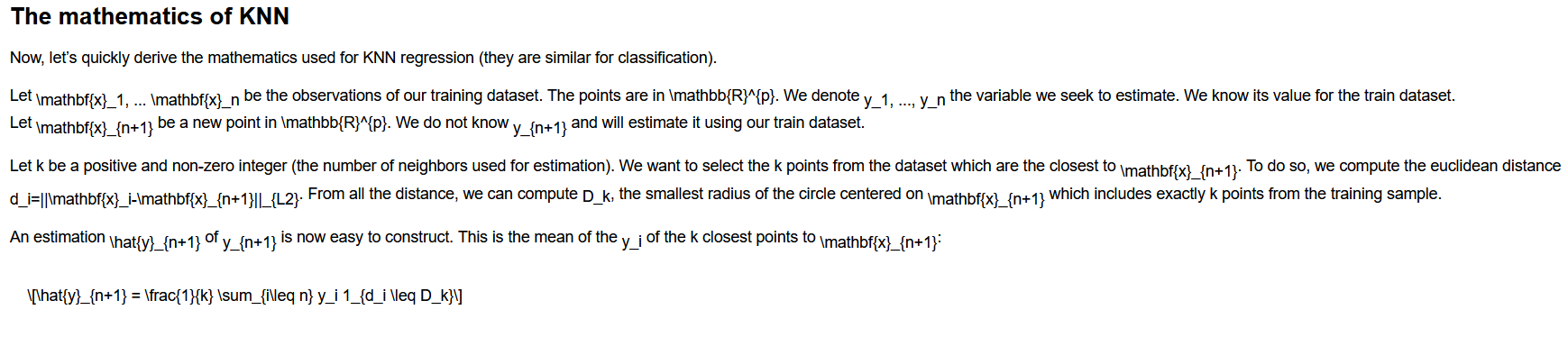
This is this second post of the “Create your Machine Learning library from scratch with R !” series. Today, we will see how you can implement K nearest neighbors (KNN) using only the linear algebra available in R. Previously, we managed to implement [PCA](http://enhancedatascience.com/2018/01/30/your-own-machine-learning-library-from-scratch-with-r/) and next time we will deal with SVM and decision trees.

The K-nearest neighbors (KNN) is a simple yet efficient classification and regression algorithm. KNN assumes that an observation will be similar to its K closest neighbors. For instance, if most of the neighbors of a given point belongs to a given class, it seems reasonable to assume that the point will belong to the same given class.

**The mathematics of KNN**

:

**KNN regression in R**

First, we build a “my\_knn\_regressor” object which stores all the training points, the value of the target variable and the number of neighbors to use.

my\_knn\_regressor <- function(x,y,k=5)

{

if (!is.matrix(x))

{

x <- as.matrix(x)

}

if (!is.matrix(y))

{

y <- as.matrix(y)

}

my\_knn <- list()

my\_knn[['points']] <- x

my\_knn[['value']] <- y

my\_knn[['k']] <- k

attr(my\_knn, "class") <- "my\_knn\_regressor"

return(my\_knn)

}

The tricky part of KNN is to compute efficiently the distance. We will use the function we created in our [previous post on vectorization](http://enhancedatascience.com/2018/05/06/machine-learning-explained-vectorization-and-matrix-operations/). The function and mathematical derivations are specified in this post.

gramMatrix <- function(X,Y)

{

tcrossprod(X, Y)

}

compute\_pairwise\_distance=function(X,Y)

{

xn <- rowSums(X \*\* 2)

yn <- rowSums(Y \*\* 2)

outer(xn, yn,'+') - 2 \* tcrossprod(X, Y)

}

Now we can build our predictor:

predict.my\_knn\_regressor <- function(my\_knn,x)

{

if (!is.matrix(x))

{

x=as.matrix(x)

}

##Compute pairwise distance

dist\_pair <- compute\_pairwise\_distance(x,my\_knn[['points']])

##as.matrix(apply(dist\_pair,2,order) <= my\_knn[['k']]) orders the points by distance and select the k-closest points

##The M[i,j]=1 if x\_j is on the k closest point to x\_i

t(as.matrix(apply(dist\_pair,2,order) <= my\_knn[['k']])) %\*% my\_knn[['value']] / my\_knn[['k']]

}

The last line may seem complicated:

1. apply(dist\_pair,2,order) orders the points by distance
2. apply(dist\_pair,2,order)<=my\_knn[['k']] selects the k-closest points to each point in our new dataset
3. M=t(as.matrix(apply(dist\_pair,2,order) <= my\_knn[['k']])) cast the matrix into a one hot matrix. if is one of the k closest points to . is zero otherwise.
4. M %\*% my\_knn[['value']] / my\_knn sums the value of the k closest points and normalises it by k

**KNN Binary Classification in R**

The previous code can be reused as it is for binary classification. Your outcome should be encoded as a one-hot variable. If the estimated output is greater (resp. less) than 0.5, you can assume that your point belongs to the class encoded as one (resp. zero). We will use the classical Iris dataset and classify the setosa versus the virginica specy.

iris\_class <- iris[iris[["Species"]]!="versicolor",]

iris\_class[["Species"]] <- as.numeric(iris\_class[["Species"]]!="setosa")

knn\_class <- my\_knn\_regressor(iris\_class[,1:2],iris\_class[,5])

predict(knn\_class,iris\_class[,1:2])

Since, we only used 2 variables, we can easily plot the decision boundaries on a 2D plot.

#Build grid

x\_coord <- seq(min(iris\_class[,1]) - 0.2,max(iris\_class[,1]) + 0.2,length.out = 200)

y\_coord <- seq(min(iris\_class[,2])- 0.2,max(iris\_class[,2]) + 0.2 , length.out = 200)

coord <- expand.grid(x=x\_coord, y=y\_coord)

#predict probabilities

coord[['prob']] <- predict(knn\_class,coord[,1:2])

library(ggplot2)

ggplot() +

##Ad tiles according to probabilities

geom\_tile(data=coord,mapping=aes(x, y, fill=prob)) + scale\_fill\_gradient(low = "lightblue", high = "red") +

##add points

geom\_point(data=iris\_class,mapping=aes(Sepal.Length,Sepal.Width, shape=Species),size=3 ) +

#add the labels to the plots

xlab('Sepal length') + ylab('Sepal width') + ggtitle('Decision boundaries of KNN')+

#remove grey border from the tile

scale\_x\_continuous(expand=c(0,0)) + scale\_y\_continuous(expand=c(0,0))

And this gives us this cool plot:

**Possible extensions**

Our current KNN is basic, but you can improve and test it in several ways:

* What is the influence of the number of neighbors ? (You should see some overfitting/underfitting)