Handmade KNN

Álvaro Orgaz Expósito

THEORY

The aim of the *K Nearest Neighbours* algorithm is to predict a target variable y^{test} (categorical for *classification* problem or continuous for *regression problem*) in an unseen *test* data comparing its associated known features $x^{test} = \{x_1^{test}, \dots, x_p^{test}\}$ with the features of a *train* data $x^{train} = \{x_1^{train}, \dots, x_p^{train}\}$ while using the known y^{train} values as a reference for the y^{test} prediction.

Firstly, let's see the main steps of this simple *machine learning* algorithm.

1. Define the inputs $x^{train}_{[NxP]}$, $y^{train}_{[Nx1]}$, $x^{test}_{[MxP]}$ and K.
2. For each M observation in the *test* data $x^{test}_{[MxP]}$, compute the mathematical distance (*Euclidean* in this paper) to each N observation in the *train* data $x^{train}_{[NxP]}$.
3. For each M observation in the *test* data $x^{test}_{[MxP]}$, select the K nearest (lower mathematical distance) observations in the *train* data $x^{train}_{[NxP]}$.
4. For each M observation in the *test* data $x^{test}_{[MxP]}$, use the found neighbours for:

   4.1 In a *regression* problem, compute the prediction $\hat{y}^{test} = mean(y_1^{train}, \dots, y_K^{train})$.

   4.2 In a *classification* problem, compute the prediction $\hat{y}^{test} = mode(y_1^{train}, \dots, y_K^{train})$.

CODE

The data used is the popular dataset *iris*. Let's predict the categorical variable *Species* (then it is a *classification* problem with 3 categories) with the explanatory features: *Petal.Length* and *Petal.Width*.

Note: It is important to *normalize* (in this paper each feature observation minus the feature minimum value divided by the feature range) the features so that their units do not affect the mathematical distance. For example, if you compute the mathematical *Euclidean* distance from a person $A = (1.80m, 70kg)$ and another $B = (1.9m, 80kg)$ the result is

$$distance(A, B) = \sqrt{(1.8 - 1.9)^2 + (70 - 80)^2} = 10$$

But if you change the units to $A = (180cm, 70000g)$ and $B = (190cm, 80000g)$ the result is

$$distance(A, B) = \sqrt{(180 - 190)^2 + (70000 - 80000)^2} = 10000$$

Let's define the inputs $x^{train}_{[NxP]}$ and $y^{train}_{[Nx1]}$. *Note*: The function *duplicated* is used to delete the observations with duplicated values in the needed variables.

```r
duplicates <- duplicated(iris[,c("Petal.Length","Petal.Width")])
X_train <- iris[!duplicates,c("Petal.Length","Petal.Width")]
Y_train <- iris[!duplicates,"Species"]
# Normalize the data
range_X1 <- max(X_train[,1])-min(X_train[,1])
range_X2 <- max(X_train[,2])-min(X_train[,2])
X_train[,1] <- (X_train[,1]-min(X_train[,1]))/range_X1
X_train[,2] <- (X_train[,2]-min(X_train[,2]))/range_X2
```

Let's define $x^{test}_{[MxP]}$ by creating artificial data with all combinations of both features from 0 to 1 by 0.01, it will be interesting for observing the *KNN decision boundaries* in the following plots. Remember that the range of our *normalized* features goes from 0 to 1.

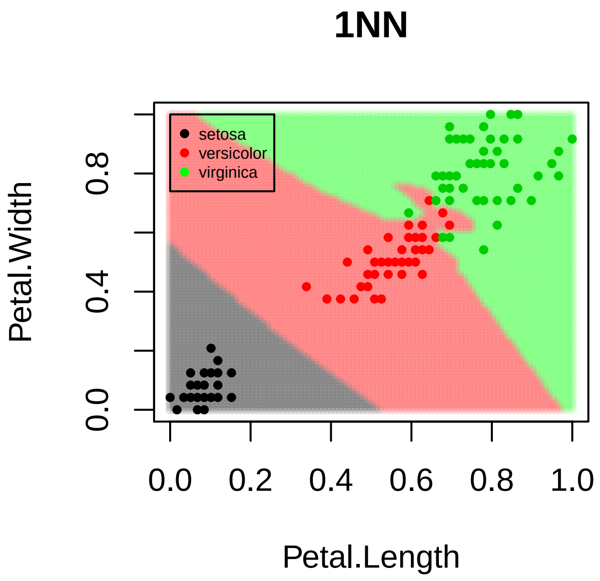
```r
X_test <- expand.grid(list(Petal.Length=seq(0,1,0.01),Petal.Width=seq(0,1,0.01)))
```

Let's create the *KNN* algorithm function.

```r
KNN <- function(X_train,X_test,Y_train,K){
  # Define the function for mathematical Euclidean distance between observations
  euclidean <- function(a,b){return(sqrt(sum((a-b)^2)))}
  # Iterate test observations
  Y_test <- c()
  for(i in 1:nrow(X_test)){
    # Iterate train observations and compute all distances to the test observation i
    distances <- c()
    for(j in 1:nrow(X_train)){
      distances <- c(distances,euclidean(X_train[j,],X_test[i,]))
    }
    # Select K neighbours and compute the prediction or mode for test observation i
    Y_neighbours <- Y_train[order(distances)][1:K]
    Y_test[i] <- names(which.max(table(Y_neighbours)))
  }
  return(Y_test)
}
```

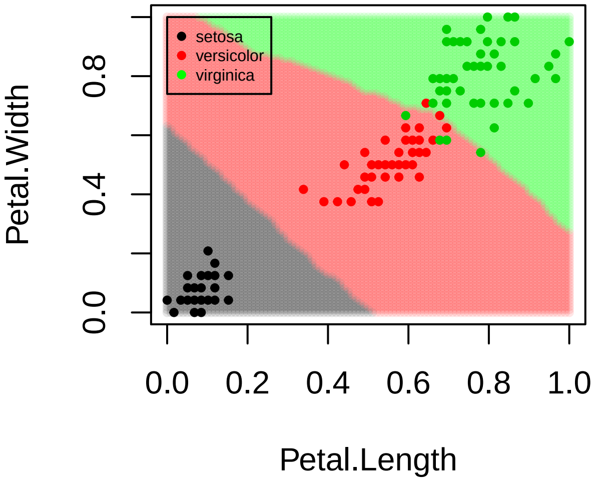
Now, let's apply the *KNN* function with $K = 1$ and plot the result.

```r
Y_test <- KNN(X_train,X_test,Y_train,K=1)
transparent_colors <- scales::alpha(c("black","red","green"),0.1)
plot(X_test[,1],X_test[,2],col=transparent_colors[as.numeric(factor(Y_test))],
     pch=19,cex=0.5,xlab="Petal.Length",ylab="Petal.Width",main="1NN")
points(X_train[,1],X_train[,2],col=Y_train,pch=19,cex=0.5)
legend(0,1,legend=unique(Y_train),col=c("black","red","green"),pch=19,cex=0.5)
```



Now, let's apply the *KNN* function with $K = 11$ and plot the result.

```
Y_test <- KNN(X_train,X_test,Y_train,K=11)
transparent_colors <- scales::alpha(c("black","red","green"),0.1)
plot(X_test[,1],X_test[,2],col=transparent_colors[as.numeric(factor(Y_test))],
     pch=19,cex=0.5,xlab="Petal.Length",ylab="Petal.Width")
points(X_train[,1],X_train[,2],col=Y_train,pch=19,cex=0.5)
legend(0,1,legend=unique(Y_train),col=c("black","red","green"),pch=19,cex=0.5)
```



In conclusion, in these plots we can observe the predictions for $x^{test}_{[MxP]}$ (transparent coloured points) and the real values $y^{train}_{[Nx1]}$ (solid coloured points). We can observe that the *decision boundary* is more flexible or complex in *1NN* than in *11NN*, and it means that the *bias* of predictions is lower but the *variance* is higher. The idea is that averaging more neighbours gives a prediction with better generalization at the expense of a *bias* increase, which can be a positive thing in *test* data for avoiding *overfitting*.