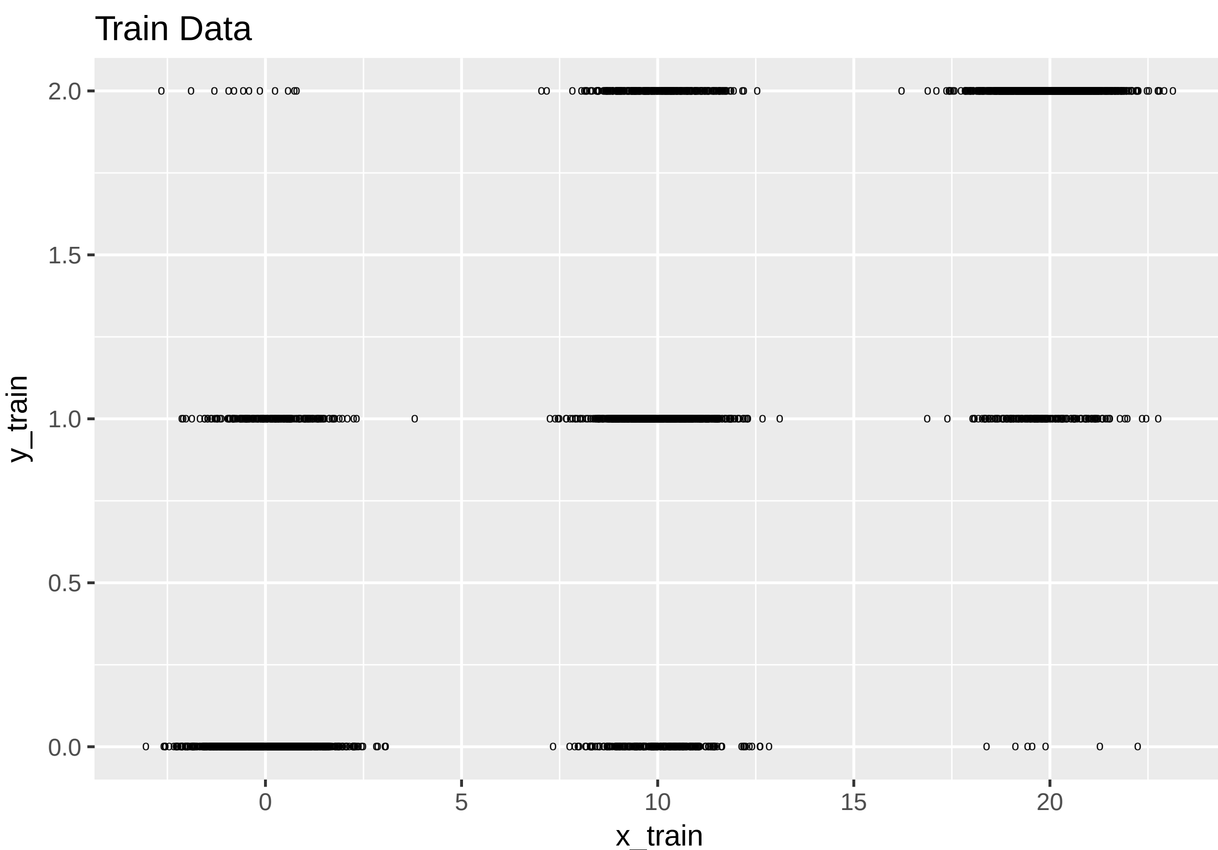
Original KNN Algorithm

```r
#y_train needs to be dummmy coded such that y = 0/1/.../nclass
#X_train and X_test can be matrices (p>1) or vectors(p=1)
KNN = function(y_train, X_train, X_test, K, nclass){

    p = length(X_train[1,]) #number of predictors
    n_train = length(y_train) #length of training data
    n_test = length(X_test[,1]) #length of test data

    y_pred_test = c()

    for(i in 1:n_test){

        x0 = X_test[i,] #ith test predictor value

        #we duplicate x0 n_train times so we can find the distance
        #from x0 to every training data point
        test_mat = matrix(x0, n_train, p, byrow = T) #
        value_diff = sqrt(rowSums((test_mat - X_train)^2))
        value = cbind(value_diff, y_train)

        #we order the training responses from low ||x0-xi|| distance
        #to high ||x0-xi|| distance
        closest = value[order(value[,1]),]

        y_N0 = closest[1:K,2] #response of the k-nearest neighbor of x0

        #we compute the proportion of response equal to each class...
        s = c(); for(l in 0:(nclass-1)){s = c(s, sum(y_N0==l))}
        props = s/K

        #... and we store the response class that has the highest proportion
        y_pred_test = c(y_pred_test, which.max(props)-1)
    }

    #y_pred_test contains the test response predicted by the method
    return(y_pred_test)
}
```

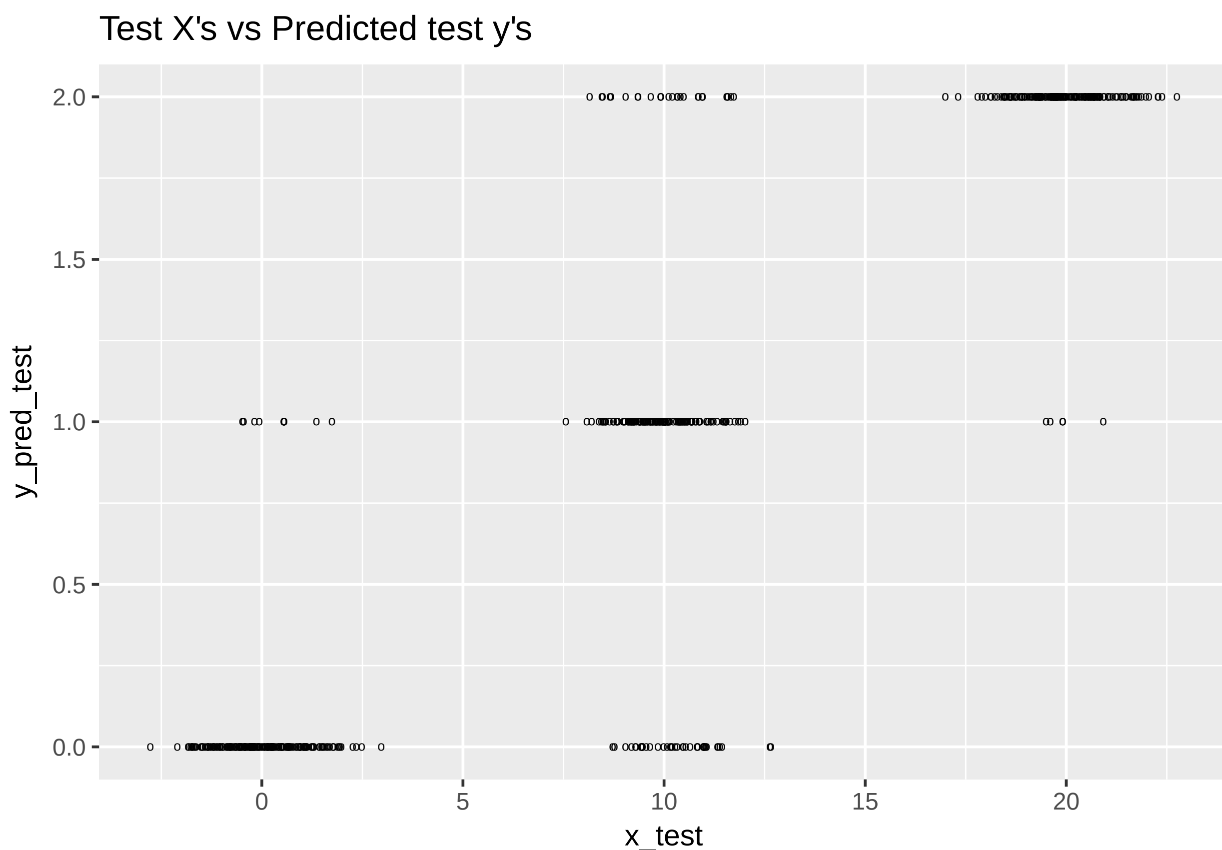
The following is a test of the KNN function with some simulated data. We build an artificial relationship between training predictors and training responses. Training x's with low mean will be assiciated with binomial responses with low proba. Training x's with middle mean will be assiciated with binomial responses with middle proba. Training x's with high mean will be assiciated with binomial responses with high proba.

```r
#simulating data

n_train = 1000; n_test = 200; n_class = 3

y_train1 = rbinom(n_train, n_class-1, .1)
y_train2 = rbinom(n_train, n_class-1, .5)
y_train3 = rbinom(n_train, n_class-1, .9)
y_train = c(y_train1, y_train2, y_train3)
```

```
x_train1 = rnorm(n=n_train, mean=0, sd=1)
x_train2 = rnorm(n=n_train, mean=10, sd=1)
x_train3 = rnorm(n=n_train, mean=20, sd=1)
x_train = as.matrix(c(x_train1, x_train2, x_train3))

#we generate test predictors the same way we generated training predictors
#with same parameters
x_test1 = rnorm(n=n_test, mean=0, sd=1)
x_test2 = rnorm(n=n_test, mean=10, sd=1)
x_test3 = rnorm(n=n_test, mean=20, sd=1)
x_test = as.matrix(c(x_test1, x_test2, x_test3))
shuff_ind = sample(1:length(x_test), replace=F)
x_test = as.matrix(x_test[shuff_ind])

df = data.frame(x_train, y_train)
ggplot(df)+ geom_point(aes(x=x_train, y=y_train), shape="o")+ggtitle("Train Data")
```




```
# source("C:/Users/Louis/Documents/UPMC/M1/Summer 2018/531T-B/assignment 2/KNN_fct.r")
y_pred_test = KNN(y_train, x_train, x_test, K=5, nclass=n_class)

df = data.frame(x_test, y_pred_test)
ggplot(df)+ geom_point(aes(x=x_test, y=y_pred_test), shape="o")+ggtitle("Test X's vs Predicted test y's
```




```r
#testing if we got parameters close from the one used above
mat = cbind(shuff_ind, y_pred_test)
ord_y_test = mat[order(mat[,1]),]
ord_y_test = ord_y_test[,2]


mean(ord_y_test[1:n_test])/(n_class-1) # should be little lower than .1
```

```
## [1] 0.025
```

```r
mean(ord_y_test[(n_test+1):(2*n_test)])/(n_class-1) #should be about .5
```

```
## [1] 0.46
```

```r
mean(ord_y_test[(2*n_test + 1):(3*n_test)])/(n_class-1) #should be a little greater than .9
```

```
## [1] 0.9875
```