Milestone 1

Andres Calderon
acald013@ucr.edu

January 23, 2016

1 Introduction

This report describes the initial steps in order to accomplish the final project in the course. The main goal of the project is to perform a reliability analysis of a machine learning algorithm (kNN). During this first milestone it is intedend to provide a flexible implementation of the kNN algorithm, a mechanism to inject random errors in the calculation of the distance metric and a brief analisys of the impact of an unreliable distance calculation.

2 A flexible implementation of kNN

There are many open source implementations of the kNN algorithm under different programming languages. This report uses the R Project for Statistical Computing[1] platform using especifically the knnflex[2] package. The knnflex package allows a more flexible implementation of the distance metric as well the oportunity to code custom functions for aggregations and tie handlers. In addition, it uses the caret package to compute the confusion matrix and associated statistics for the model fit.

2.1 A quick classification example

The code in figure 1 illustrates the use of knnflex to classify a small random set of features. In lines 5 to 10 it sets the number of instances and a random seed, create two attributes with random numbers (x1 and x2) and a binary class (y). Lines 11 and 12 divede the data set in training and testing set (75% and 25% respectively). Line 17 call the kdd.dist function which will generate a distance matrix among all the instances in the data set. Line 18 perform the classification calling the knn.predict function. It takes the training and testing datasets, the distance matrix, the number of neighbors to be taken into account and the aggregation method to pick the class between them.

[1] https://www.r-project.org/

[2] `http://ftp.uni-bayreuth.de/math/statlib/R/CRAN/src/contrib/Descriptions/knnflex.html`

```
1  require(knnflex)
2  require(caret)
3
4  # a quick classification example
5  n <- 200
6  set.seed(123)
7  x1 <- c(rnorm(n/2,mean=2.5),rnorm(n/2,mean=7.5))
8  x2 <- c(rnorm(n/2,mean=7.5),rnorm(n/2,mean=2.5))
9  x  <- cbind(x1,x2)
10 y <- c(rep(1,n/2),rep(0,n/2))
11 train <- sample(1:n,n*0.75)
12 test <- (1:n)[-train]
13 # plot the training cases
14 plot(x1[train],x2[train],col=y[train]+1,xlab="x1",ylab="x2"
15      ,xlim=c(-1,10),ylim=c(-1,10))
16 # predict the other cases
17 kdist <- knn.dist(x)
18 preds <- knn.predict(train,test,y,kdist,k=3,agg.meth="majority")
19 # add the predictions to the plot
20 points(x1[test],x2[test],col=as.integer(preds)+1,pch="+")
21 # display the confusion matrix
22 confusionMatrix(y[test],preds)
```

Figure 1: A quick code example

Finally, line 22 calls the confusionMatrix function to retrieve the accuracy and other statistics from the model (figure 2). Lines 14 and 20 plots the initial instances in the training set and the result of the classification for the instances in the testing set. Figures show the results respectively.

```
confusionMatrix(y[test],preds)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 27  0
##          1  0 23
##
##                Accuracy : 1
##                  95% CI : (0.9289, 1)
##     No Information Rate : 0.54
##     P-Value [Acc > NIR] : 4.166e-14
##
##                   Kappa : 1
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.00
##             Specificity : 1.00
##          Pos Pred Value : 1.00
##          Neg Pred Value : 1.00
##              Prevalence : 0.54
##          Detection Rate : 0.54
##    Detection Prevalence : 0.54
##       Balanced Accuracy : 1.00
##
##        'Positive' Class : 0
##
```

Figure 2: Confusion matrix

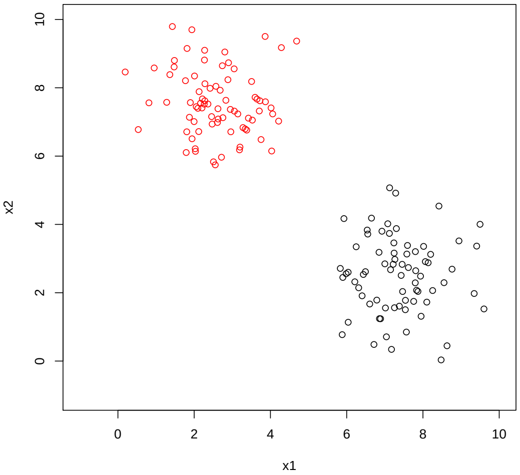

Figure 3: Instances in training set.

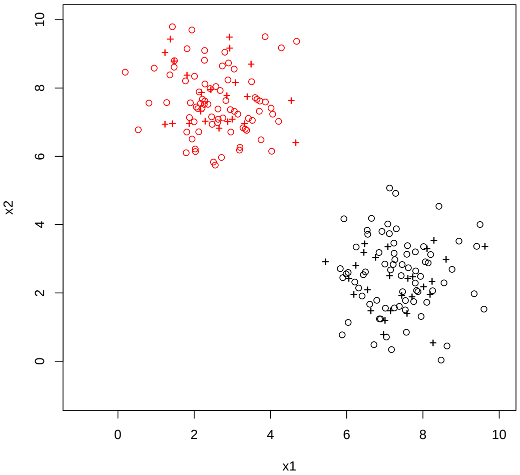

Figure 4: Results for instances in testing set.