



K-NN implementation in R

## $k$ -NN Implementation in R

- $k$ -NN classification algorithm is implemented in the “Class” package.
- The `knn()` function identifies the  $k$ -nearest neighbors using Euclidean distance where  $k$  is a user-specified number.
- The function `knn()` takes the training and test set at the same time and return the predicted class values for the test dataset.
- Numeric attributes should be normalized before passing to the function.

The  $k$ -NN algorithm is implemented as part of many packages, including Class, and FNN. Here, we will see its implementation as part of the Class package. Again, the code for this available on Github.

The slide features a blue header with the Kent State University logo and the title "GitHub". A bullet point states that all code is available on GitHub at a specific URL. The slide has a decorative background with diagonal lines and a footer with the university's website and a small graphic of yellow stars.


**KENT STATE UNIVERSITY**

## GitHub

- **All code for this module and other modules are available at <https://github.com/KSU-MSBA/64060.git>**

[WWW.KENT.EDU](http://WWW.KENT.EDU)

A reminder that all code for this module and all other modules are available on the github account.



## k-NN Implementation in R: Example

```

library(class) #install first if needed
library(caret)
library(ISLR)
summary(Default)


## default      student      balance      income
## No :9667      No :7056      Min.   : 0.0      Min.   : 772
## Yes: 333      Yes:2944      1st Qu.: 481.7    1st Qu.:21340
##                               Median : 823.6    Median :34553
##                               Mean   : 835.4    Mean   :33517
##                               3rd Qu.:1166.3    3rd Qu.:43808
##                               Max.    :2654.3    Max.    :73554

#normalize the data first: build a model and apply
norm_model<-preProcess(Default, method = c('range'))
Default_normalized<-predict(norm_model,Default)

#Let's use income and balance to predict (dropping the second variable student status)
Default_normalized<-Default_normalized[,-2]


# Use 80% of data for training and the rest for testing
Index_Train<-createDataPartition(Default_normalized$default, p=0.8, list=FALSE)
Train <-Default_normalized[Index_Train,]
Test  <-Default_normalized[-Index_Train,]

```



[WWW.KENT.EDU](http://WWW.KENT.EDU)

Again, remember to install the package as necessary. The data should be normalized, and split into training and test before invoking the algorithm.



## Effect of $k$ in the $k$ -NN classifier


```
# First column is the default status (i.e. class output)
# Second and third columns are normalized "balance" and "income"
Train_Predictors<-Train[,2:3]
Test_Predictors<-Test[,2:3]

Train_labels <-Train[,1]
Test_labels  <-Test[,1]

# train a knn model where k=4
Predicted_Test_labels <-knn(Train_Predictors,
                           Test_Predictors,
                           cl=Train_labels,
                           k=4 )

# Look at the 6 first values of predicted class (i.e., default status) of test set
head(Predicted_Test_labels)

## [1] No No No No No No
## Levels: No Yes
```



WWW.KENT.EDU

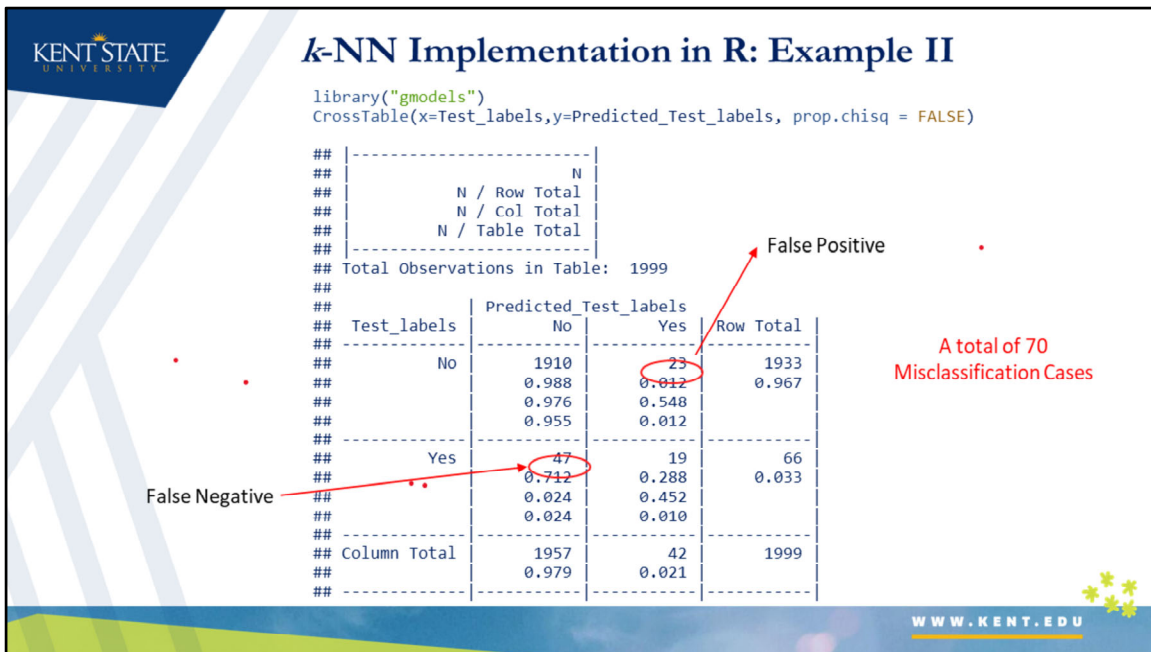
Here, we only use balance and income as predictors. The response is the Default status.

## Confusion Matrix

- As we discussed, confusion matrix is a two by two table that contains four outcomes produced by a binary classifier.
- A Confusion Matrix can be used to create the many different measures of a model's fitness including: accuracy, precision and recall.
- `CrossTable()` function that is part of the package 'gmodels' can be used to compile an informative confusion matrix.

A confusion matrix is a cross-table that provides us with the details needed to compute several outcomes, including how many observations were correctly classified (accuracy), true positive rate (recall or sensitivity), the positive predictive value (precision), or specificity (false negative). We can easily generate this using the `CrossTable()` function in the gmodels package.

An algorithm with high recall is able to predict with high accuracy the positive or relevant values, while high precision indicates what proportion of positives that are identified are truly positive



The table provides the confusion matrix for our model. A total of 70 cases were misclassified. Note that the confusion matrix can be used to calculate further measures. Specifically, can you calculate the following:

1. Accuracy = Ratio of (TP + TN) / N
2. Recall (sensitivity) = TP / (TP + FN)
3. Precision (positive predictive value) = TP / (TP + FP)
4. Specificity (true negative rate) = TN / (TN + FP)

## Probabilities

- Sometimes, it is preferred to have raw prediction probabilities rather than predicted class labels.
- `knn()` can additionally return probabilities if 'prob' argument is set to true.
- The probability is defined as the proportion of nearest neighbors that belongs to the majority class
- For example, the probability will be 0.6 if 3 out of 5 nearest neighbors are from one class.

```
Predicted_Test_labels <- knn(Train_Predictors,  
                             Test_Predictors,  
                             cl=Train_labels, k=100, prob=TRUE )  
  
class_prob <- attr(Predicted_Test_labels, 'prob')  
head(class_prob)  
  
## [1] 1.00 1.00 0.99 1.00 0.99 1.00
```

Sometimes it is useful to know what proportion of neighbors were of the majority class. The higher the proportion, the greater the confidence that the observation belongs to that class. This can be easily determined by the prob argument.