Classification Using k-Nearest Neighbors

(Project Summary)

Ami Patel

CS 4375.501

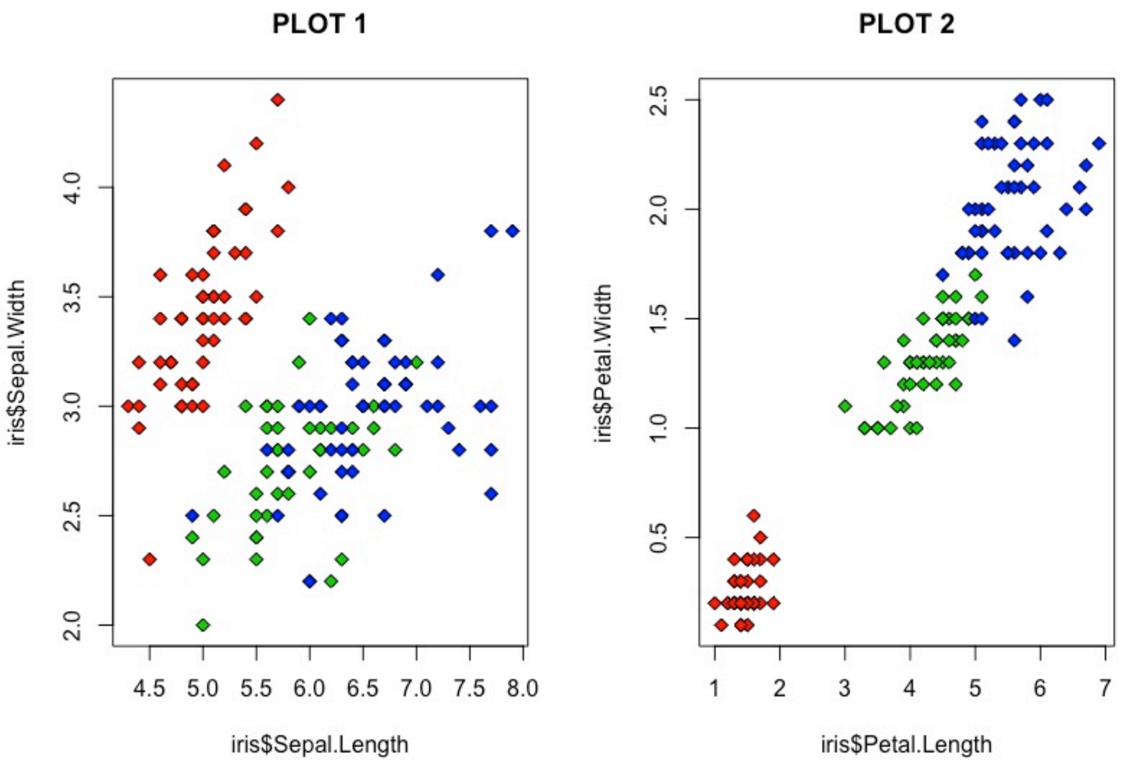
Dr. Karen Mazidi

March 3, 2019

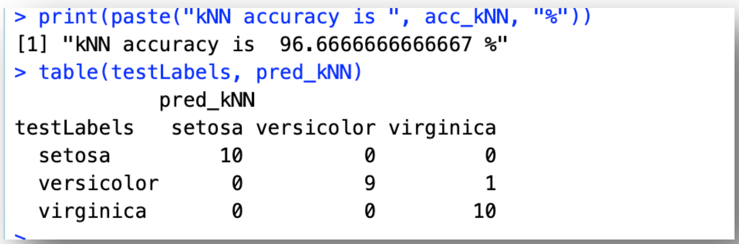
The k-Nearest Neighbors (kNN) algorithm is a non-parametric supervised learning algorithm, that does not "model" the data but instead it is highly based on the concept of "instance-based" or lazy- learning. kNN algorithm is often used to perform multi-class classification because it is easier and less complex when compared to the "one vs all" approach by logistic regression.

R Script:

For the project, the data used was **iris** data, which is a built-in R dataset by default. After the data exploration of **iris** data, it can be found the data frame has 150 instances and 5 variables. These 5 variables are namely, *Sepal.Length, Sepal.Width, Petal.Length, Petal.Width and Species*. The variable *Species* has three levels, **"setosa"**, **"versicolor"** and **"virginica,"** with 50 observations in each level. To get a rough idea of how these species are divided based on Sepal.Length and Sepal.Width, a plot was made with **setosa** being red, **versicolor** being green and **virginica** being blue. Also, a second plot denoting separation of species based on Petal.Length and Petal.Width was made using the same color-coding scheme.

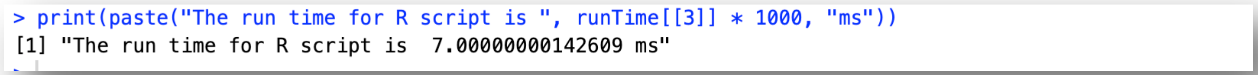


The algorithm in R script, as required, was timed using proc.time () and startTime was recorded after the aforementioned plot was made. The iris data was divided into two sets, 80% of the data in train and the remaining 20% of the data in test and knn () from library (class) was used to classify the test data based on learning from train data. knn () performed the classification and returned a factor with 3 levels as a result. The results were then compared to the actual *test$Species* values to learn the accuracy of knn () model. To get a clear idea of the results, a confusion matrix was printed to know the model better.



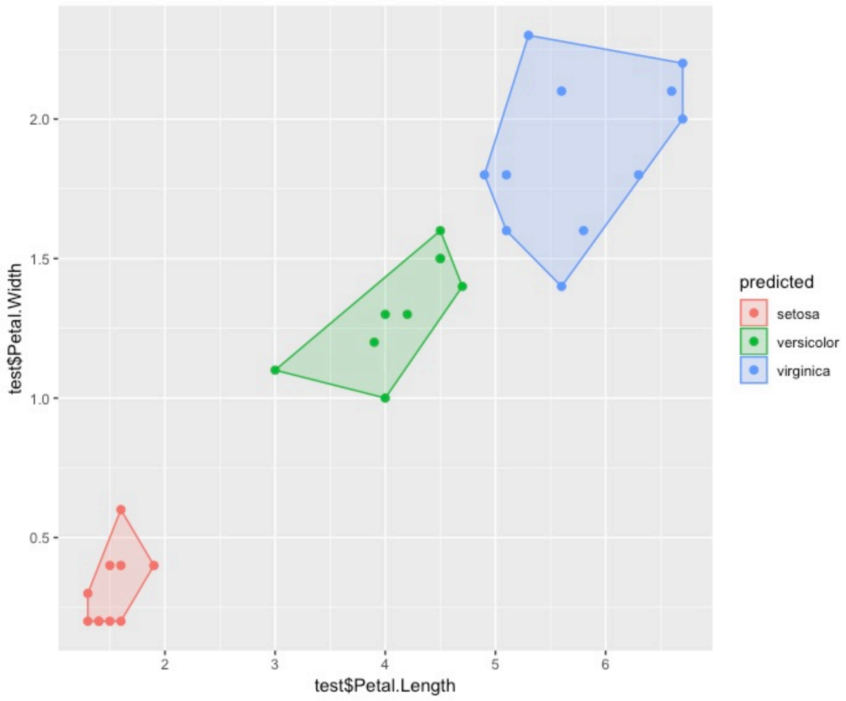
```
> print(paste("kNN accuracy is ", acc_kNN, "%"))
[1] "kNN accuracy is  96.6666666666667 %"
> table(testLabels, pred_kNN)
            pred_kNN
testLabels   setosa versicolor virginica
  setosa         10          0         0
  versicolor      0          9         1
  virginica       0          0        10
```

Once the confusion matrix was printed, endTime of the algorithm was recorded using proc.time() and the difference of (endTime – startTime) was calculated and displayed at the console to get the run-time of kNN algorithm in R script.



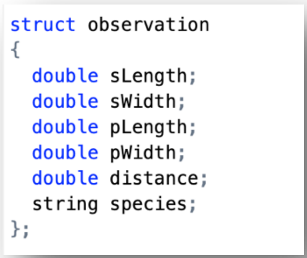
```
> print(paste("The run time for R script is ", runTime[[3]] * 1000, "ms"))
[1] "The run time for R script is  7.00000000142609 ms"
```

Along with knowing the above metrics, a plot was made for the predicted values showing different decision boundaries for different species. The plot shows clearly the test values with red in color were classified by knn () as **setosa**, the ones with green in color were classified as **versicolor** and the blue ones were classified as **virginica**.



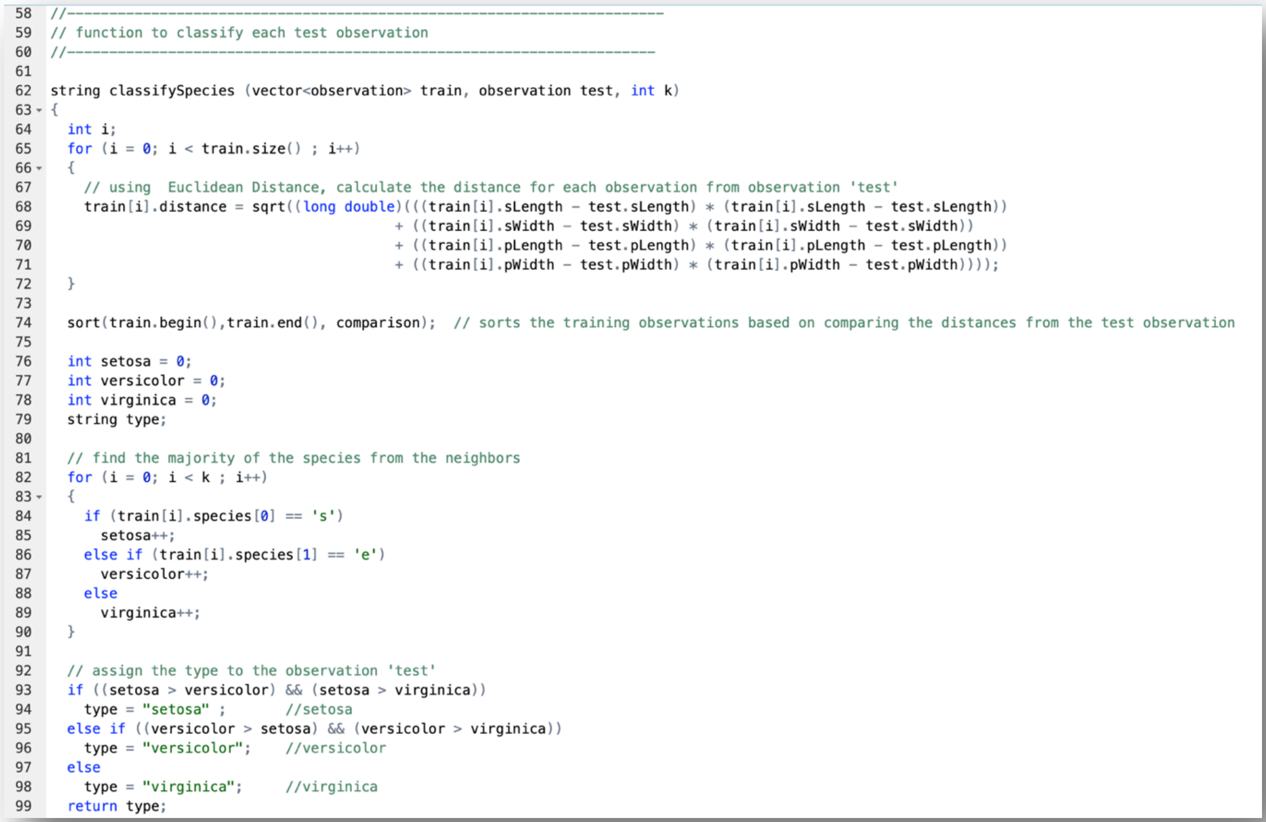
C++ Script:

Every observation was made as an object of type **struct observation**, which helped holding all the data for a specific observation together. The run time was calculated using steady_clock and duration under **#include <chrono>** header. Every observation from the test data was passed to a function called **classifySpecies ()**, which performed the following tasks in order and implemented **kNN algorithm**:



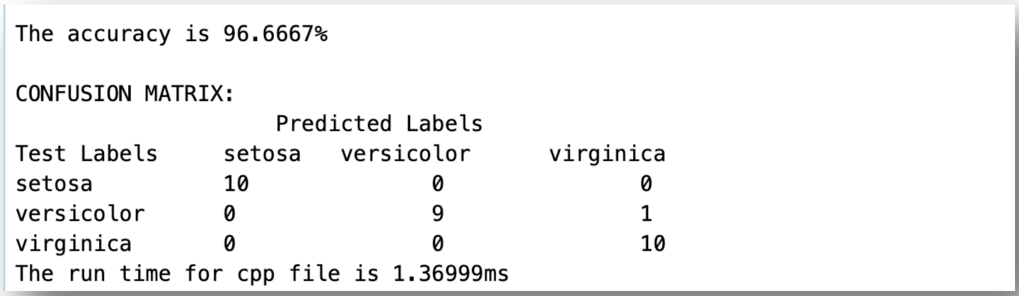
```
struct observation
{
  double sLength;
  double sWidth;
  double pLength;
  double pWidth;
  double distance;
  string species;
};
```

1. Go through all the train observations/points and compute the distance of each training observation from the test observation using Euclidean Distance formula.
2. Sort the train vector by comparing the distance from the test observation.
3. Go through the first k observations after sorting and make a tally on whether the neighbor is setosa (notice that the first alphabet is 's'), versicolor (notice, that the second alphabet is 'e') or virginica. Comparing using single alphabets is faster than comparing whole strings.
4. Pick a majority of all the counts and return the predicted type of the test observation.



```
//-----------------------------------------------------------------------
// function to classify each test observation
//----------------------------------------------------------------------

string classifySpecies (vector<observation> train, observation test, int k)
{
  int i;
  for (i = 0; i < train.size() ; i++)
  {
    // using  Euclidean Distance, calculate the distance for each observation from observation 'test'
    train[i].distance = sqrt((long double)(((train[i].sLength - test.sLength) * (train[i].sLength - test.sLength))
                                         + ((train[i].sWidth - test.sWidth) * (train[i].sWidth - test.sWidth))
                                         + ((train[i].pLength - test.pLength) * (train[i].pLength - test.pLength))
                                         + ((train[i].pWidth - test.pWidth) * (train[i].pWidth - test.pWidth))));
  }

  sort(train.begin(),train.end(), comparison);  // sorts the training observations based on comparing the distances from the test observation

  int setosa = 0;
  int versicolor = 0;
  int virginica = 0;
  string type;

  // find the majority of the species from the neighbors
  for (i = 0; i < k ; i++)
  {
    if (train[i].species[0] == 's')
      setosa++;
    else if (train[i].species[1] == 'e')
      versicolor++;
    else
      virginica++;
  }

  // assign the type to the observation 'test'
  if ((setosa > versicolor) && (setosa > virginica))
    type = "setosa" ;        //setosa
  else if ((versicolor > setosa) && (versicolor > virginica))
    type = "versicolor";     //versicolor
  else
    type = "virginica";      //virginica
  return type;
```

A counter was set to know the accuracy of the classifications made by kNN. Also, a confusion matrix was printed along in C++ to compare the results with confusion matrix by R script.



```
The accuracy is 96.6667%

CONFUSION MATRIX:
                    Predicted Labels
Test Labels     setosa   versicolor      virginica
setosa          10             0              0
versicolor      0              9              1
virginica       0              0              10
The run time for cpp file is 1.36999ms
```

<u>Conclusion:</u>

If we notice carefully, it can be seen that the accuracy for kNN algorithm in R and C++ was the same, 96.667%. In addition, both R script and .cpp file resulted in same confusion matrix for the predicted values. This unusual behavior could be due to the limited training data for this dataset. However, the results of this algorithm could be improved further by incorporating more data or using cross-validation technique.

On the contrary, the run times for kNN algorithm in R and C++ differ significantly. The run time for the algorithm in R was found to be 7.00 milliseconds, while run-time for the same algorithm in C++ was found to be 1.37 milliseconds. This proves that C++ is faster in implementing the algorithm than R. This is because C++ is highly supported by different libraries to accomplish the computational tasks and so, it performs better than R when we compare run-times (Gillespie & Lovelace, 2017).

References:

Gillespie, C., & Lovelace, R. (2017, April 10). Efficient R programming. Retrieved from https://csgillespie.github.io/efficientR/performance.html#rcpp