Project1: Classification

Dataset Preparation

The dataset for the project was obtained from Wikipedia website [https://en.wikipedia.org/wiki/List_of_countries_by_life_expectancy].

I created a SpreadSheet and copied all the data from List by the CIA (2016). Manually added data for a new column 'Continent' which is used as label for training different models. This data was converted into a csv file named 'Project1Dataset.csv'. Prior to converting I removed commas that were populated inside each column so as not to mess with the csv data structure. The csv file has the following variables – Rank, Country, Overall Life (Expectancy at birth), Male Life (Expectancy at birth), Female Life (Expectancy at birth) and Continent. We take Overall Life, Male Life and Female Life as the features and Continent as the class label.

Problem Statement:
Given Overall,Male and Female Life Expectancy at birth of a particular country, classify the Continent to which the country belongs.

Train/Test datasets:
The R Script handles splitting of data into various permutations of test and train sets. The main data from the CSV file is split into 5 different sets of train and test data in the ratio of 80 to 20. Used random sampling for the split. The generated data sets are stored in 2 different lists - 'trainList' and 'testList'. I also used 5 different 'seeds' in order for the application to produce coherent data models each time when the script is run.

Classification Methods

1. Decision Tree

a. C4.5 Algorithm

C4.5 builds decision trees from a set of training data using the concept of information gain. The training data is a set of already classified samples. At each node of the tree, C4.5 chooses the attribute of the data that most effectively splits its set of samples into subsets enriched in one class or the other. The splitting criterion is the normalized information gain (difference in entropy). The attribute with the highest normalized information gain is chosen to make the decision. The C4.5 algorithm then recurs on the smaller sublists.

In this project, C4.5 algorithm is implemented using 'J48' decision tree module implemented in 'Rweka' Library. A model is trained using 5 different sets of train data. The trained model is then used for predicting the class label 'Continent' for different sets of Test data. Performance Metrics like Accuracy, Precision, Recall and F-Scores are calculated for each of the 5 test sets. A final aggregate of the reported performance metrics are calculated and stored as a list.

The parameters used in this model during training phase are the feature variables and the label variable. The feature variables are given as the formula, train data as a data frame and a weka control variable to be passed to the weka learner. Summary of the model gives data such as Confusion matrix, Correctly

Classified Instances, Incorrectly Classified Instances, Kappa statistic ,Mean absolute error ,Root mean squared and relative absolute error.

The Model is stored in variable 'c45Fit' and sample of summary(c45Fit) is shown below:

```
=== Summary ===

Correctly Classified Instances         110               61.4525 %
Incorrectly Classified Instances        69               38.5475 %
Kappa statistic                          0.5121
Mean absolute error                      0.1747
Root mean squared error                  0.2982
Relative absolute error                 64.9869 %
Root relative squared error             81.3876 %
Total Number of Instances              179

=== Confusion Matrix ===

  a  b  c  d  e  f   <-- classified as
 38  1  3  0  2  1 |  a = Africa
  3 25  7  1  1  1 |  b = Asia
  0  2 35  0  0  2 |  c = Europe
  2  9 13  4  0  1 |  d = North America
  0  6  5  1  4  0 |  e = Oceania
  0  7  1  0  0  4 |  f = South America
```

Information Gain gives an insight into how the algorithm has decided to build the decision tree. It provides the data as to how much importance each attribute has during training. Infomation Gain can be calculated in R using 'Fselector' library.

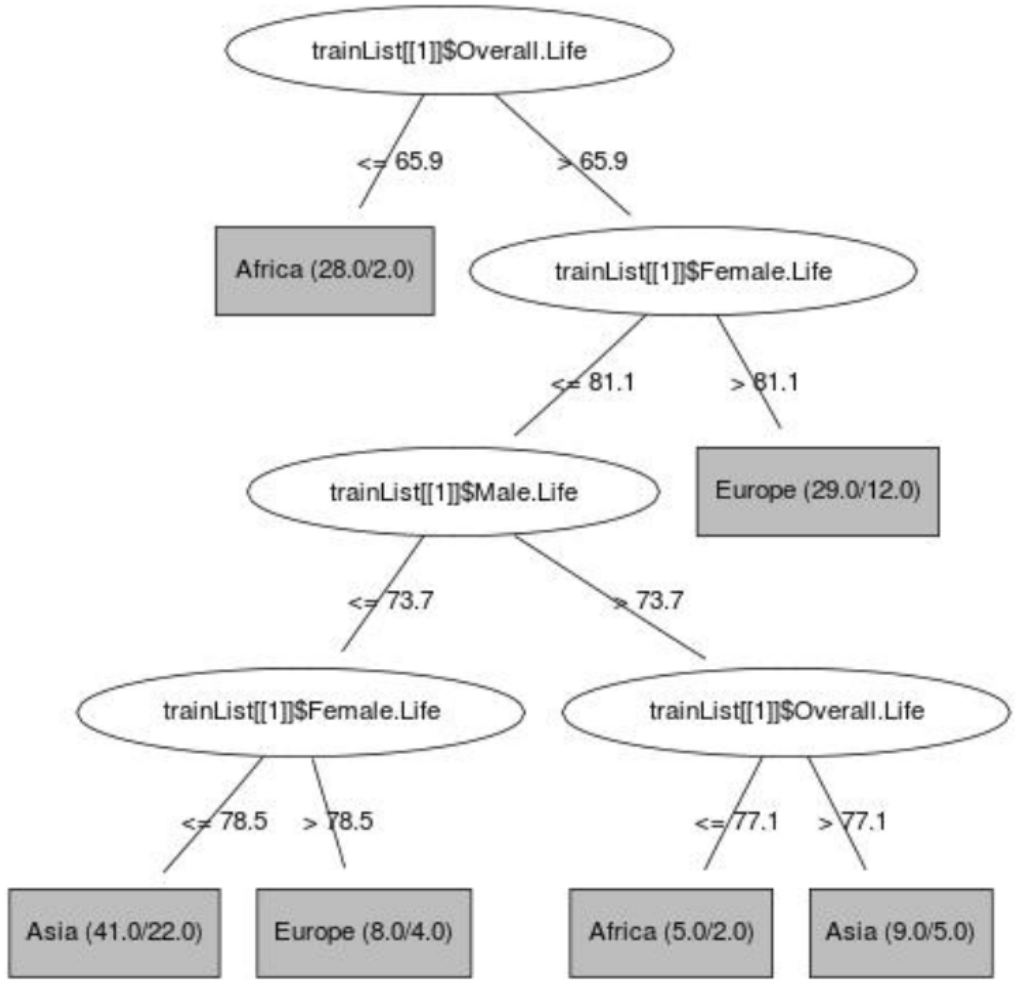
The information gain is stored in variable 'infoGainC45' and a sample is shown below:

== Information Gain ==

```
                            attr_importance
trainList[[1]]$Overall.Life        1.450592
trainList[[1]]$Male.Life           1.446644
trainList[[1]]$Female.Life         1.434229
```

Here, the feature 'Overall.Life' has more importance and hence will be used as the split criteria.
The same data can be visualized using package 'Rgraphviz' and the whole decision tree can be obtained as shown in the next page. The decision trees for each training set are saved as jpeg files when the script is executed.

Prediction for the model is done using the 'predict' function. The trained model and the test datasets are given as parameters to obtain predicted classes which can be measured using the performance metrics. The metrics are calculated using 'caret' package. Precision is calculated as positive predicted value, recall as sensitivity and F1 score is calculated using the formula
2 * ((precision * recall) / (precision + recall))



The final aggregated metrics for C45 is stored in the list ‘c45 metrics’ .

Sample c45 metric

```
$Accuracy
[1] 0.6590909

$Precision
[1] 0.8571429

$Recall
[1] 0.75

$Fscore
[1] 0.8
```

b. RIPPER Algorithm

Repeated Incremental Pruning to Produce Error Reduction (RIPPER) is a propositional rule learner. It is based in association rules with reduced error pruning (REP), a very common and effective technique found in decision tree algorithms. In REP for rules algorithms, the training data is split into a growing

set and a pruning set. First, an initial rule set is formed that over ts the growing set, using some heuristic method. This overlarge rule set is then repeatedly simplified by applying one of a set of pruning operators typical pruning operators would be to delete any single condition or any single rule. At each stage of simplification, the pruning operator chosen is the one that yields the greatest reduction of error on the pruning set. Simplification ends when applying any pruning operator would increase error on the pruning set.

In this project, RIPPER algorithm is implemented using 'JRip' decision tree module implemented in 'Caret' Library. A model is trained using 5 different sets of train data. The trained model is then used for predicting the class label 'Continent' for different sets of Test data. Performance Metrics like Accuracy, Precision, Recall and F-Scores are calculated for each of the 5 test sets. A final aggregate of the reported performance metrics are calculated and stored as a list - 'Ripper_metrics'

The parameters used in this model during training phase are the feature variables, the label variable, Jrip method, preprocessing criteria (Center and Scaling), a train control variable of method 'cv' and tune length of 10.

The Model is stored in variable 'jripFit' and sample of the execution is shown below:
Rule-Based Classifier

```
179 samples
  3 predictors
  6 classes: 'Africa', 'Asia', 'Europe', 'North America', 'Oceania', 'South America'

Pre-processing: centered (3), scaled (3)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 162, 160, 159, 162, 161, 161, ...
Resampling results across tuning parameters:

  NumOpt  NumFolds  MinWeights  Accuracy   Kappa
  1        2        1           0.4631067  0.2901454
  1        2        2           0.4571199  0.2838187
  1        2        3           0.4116477  0.2180469
  1        2        4           0.4470790  0.2662997
  1        2        5           0.4373568  0.2516875
  1        2        6           0.4170937  0.2245722
  1        2        7           0.4296345  0.2418806
  1        3        1           0.4680366  0.2987715
  1        3        2           0.4662801  0.2979815
……. [ reached getOption("max.print") -- omitted 290 rows ] ……...

Accuracy was used to select the optimal model using  the largest value.
The final values used for the model were NumOpt = 6, NumFolds = 11 and MinWeights = 5.
```

Prediction for the model is done using the 'predict' function. The trained model and the test datasets are given as parameters to obtain predicted classes which can be measured using the performance metrics. The metrics are calculated using 'caret' package. Precision is calculated as positive predicted value, recall as sensitivity and F1 score is calculated using the formula
*2 * ((precision * recall) / (precision + recall))*

The final aggregated metrics for RIPPER is stored in the list 'Ripper_metrics' .

Sample RIPPER metric:

$Accuracy

```
[1] 0.5

$Precision
[1] 0.3333333

$Recall
[1] 1

$Fscore
[1] 0.5
```

Important Note: jRip takes a longer time to get trained due to which the execution of whole script takes much longer time.

2. Support Vector Machines

The support vector machine (SVM) approach focuses on finding the optimal separation boundary between data points that have different classifications. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall. In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

In this project, Support Vector Machine is implemented using ‘svm’ module implemented in ‘e1071’ Library. A model is trained using 5 different sets of train data. The trained model is then used for predicting the class label ‘Continent’ for different sets of Test data. Performance Metrics like Accuracy, Precision, Recall and F-Scores are calculated for each of the 5 test sets. A final aggregate of the reported performance metrics are calculated and stored as a list - ‘Svm_metrics’

The parameters used in this model during training phase are the feature variables and the label variable. The feature variables are given as the formula, train data as a data frame and a weka control variable to be passed to the weka learner.

The Model is stored in variable ‘svmFit’ and sample of summary(svmFit) is shown below:

Call:

```
svm(formula = trainDf$Continent ~ trainDf$Overall.Life + trainDf$Male.Life +
trainDf$Female.Life,
    data = trainDf, control = Weka_control(R = TRUE))

Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.3333333

Number of Support Vectors:  147

 ( 32 38 16 29 20 12 )

Number of Classes:  6
```

```
Levels:
 Africa Asia Europe North America Oceania South America
```

Prediction for the model is done using the ‘predict’ function. The trained model and the test datasets are given as parameters to obtain predicted classes which can be measured using the performance metrics. The metrics are calculated using ‘caret’ package. Precision is calculated as positive predicted value, recall as sensitivity and F1 score is calculated using the formula

*2 * ((precision * recall) / (precision + recall))*

Confusion Matrix and Statistics of a SVM model:

```
                Reference
Prediction      Africa Asia Europe North America Oceania South America
  Africa             6    0      0             0       0             0
  Asia               0    5      0             0       0             0
  Europe             2    2     16             7       4             2
  North America      0    0      0             0       0             0
  Oceania            0    0      0             0       0             0
  South America      0    0      0             0       0             0

Overall Statistics

                Accuracy : 0.6136
                  95% CI : (0.455, 0.7564)
    No Information Rate : 0.3636
    P-Value [Acc > NIR] : 0.0006502

                   Kappa : 0.4355
 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: Africa Class: Asia Class: Europe Class: North America Class:
Oceania
Sensitivity                 0.7500      0.7143        1.0000               0.0000
0.00000
Specificity                 1.0000      1.0000        0.3929               1.0000
1.00000
Pos Pred Value              1.0000      1.0000        0.4848                  NaN
NaN
Neg Pred Value              0.9474      0.9487        1.0000               0.8409
0.90909
Prevalence                  0.1818      0.1591        0.3636               0.1591
0.09091
Detection Rate              0.1364      0.1136        0.3636               0.0000
0.00000
Detection Prevalence        0.1364      0.1136        0.7500               0.0000
0.00000
Balanced Accuracy           0.8750      0.8571        0.6964               0.5000
0.50000
                     Class: South America
Sensitivity                       0.00000
Specificity                       1.00000
Pos Pred Value                        NaN
Neg Pred Value                    0.95455
Prevalence                        0.04545
Detection Rate                    0.00000
```

```
Detection Prevalence             0.00000
Balanced Accuracy                0.50000
```

The final aggregated metrics for svm is stored in the list ‘Svm_metrics’ .

$Accuracy
```
[1] 0.6136364

$Precision
[1] 1

$Recall
[1] 0.75

$Fscore
[1] 0.8571429
```

3. K- Nearest Neighbors

The KNN or k-nearest neighbors algorithm is an example of instance-based learning, where new data are classified based on stored, labeled instances. More specifically, the distance between the stored data and the new instance is calculated by means of some kind of a similarity measure. In other words, the similarity to the data that was already in the system is calculated for any new data point that you input into the system. Then, we use this similarity value to perform predictive modeling. The higher the score for a certain data point that was already stored, the more likely that the new instance will receive the same classification as that of the neighbor.

In this project, k-nearest neighbor is implemented using ‘knn’ module implemented in ‘caret’ Library. A model is trained using 5 different sets of train data. The trained model is then used for predicting the class label ‘Continent’ for different sets of Test data. Performance Metrics like Accuracy, Precision, Recall and F-Scores are calculated for each of the 5 test sets. A final aggregate of the reported performance metrics are calculated and stored as a list - ‘Knn_metrics’ Usage of caret package ensures that finding optimal ‘k’ value is taken care of by the caret package.

The parameters used in this model during training phase are the feature variables, the label variable, knn method, preprocessing criteria (Center and Scaling), a train control variable of method ‘cv’ and tune length of 10.

The Model is stored in variable ‘knnFit’ and sample of the execution is shown below:

k-Nearest Neighbors

```
179 samples
  3 predictors
  6 classes: 'Africa', 'Asia', 'Europe', 'North America', 'Oceania', 'South America'

Pre-processing: centered (3), scaled (3)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 162, 161, 160, 162, 160, 160, ...
Resampling results across tuning parameters:

  k   Accuracy   Kappa
```

```
  5  0.4962719  0.3696995
  7  0.5432362  0.4289835
  9  0.5321659  0.4147910
 11  0.5485746  0.4342776
 13  0.5417054  0.4256991
 15  0.5298998  0.4093634
 17  0.5308867  0.4096829
 19  0.5357478  0.4138527
 21  0.5029648  0.3713791
 23  0.5101264  0.3799578

Accuracy was used to select the optimal model using  the largest value.
The final value used for the model was k = 11.
```

Prediction for the model is done using the 'predict' function. The trained model and the test datasets are given as parameters to obtain predicted classes which can be measured using the performance metrics. The metrics are calculated using 'caret' package. Precision is calculated as positive predicted value, recall as sensitivity and F1 score is calculated using the formula
*2 * ((precision * recall) / (precision + recall))*

The final aggregated metrics for knn is stored in the list 'Knn_metrics' .

Sample Knn_metric:

```
$Accuracy
[1] 0.4772727

$Precision
[1] 0.75

$Recall
[1] 0.75

$Fscore
[1] 0.75
```

Conclusion

Classification of LifeExpectancy Dataset is done using various methods like Decision Trees, SVM and KNN. Out of the 4 methods used, for this particular dataset the C4.5 decision tree algorithm produces highest accuracy and KNN produces least accuracy. The accuracy for each of the method can be improved by intelligent selection of train and test set (experimenting with different seeds) and by increasing the number of data samples in each set so that the trained model accuracy will be improved. Though various models have different accuracies, the performance of each of the models is almost in the range of 47 – 66%. Since number of data samples is less, this is considered to be of decent statistics.

References

http://spu.fem.uniag.sk/cvicenia/ksov/fuskova-ulicna/Data%20mining/cvicenie8_classification/Cv8_classification_some%20interpretations.pdf
https://en.wikipedia.org/wiki/List_of_countries_by_life_expectancy
http://data-mining.business-intelligence.uoc.edu/home/j48-decision-tree
https://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Classification/JRip
https://www.datacamp.com/community/tutorials/machine-learning-in-r
https://eight2late.wordpress.com/2017/02/07/a-gentle-introduction-to-support-vector-machines-using-r/