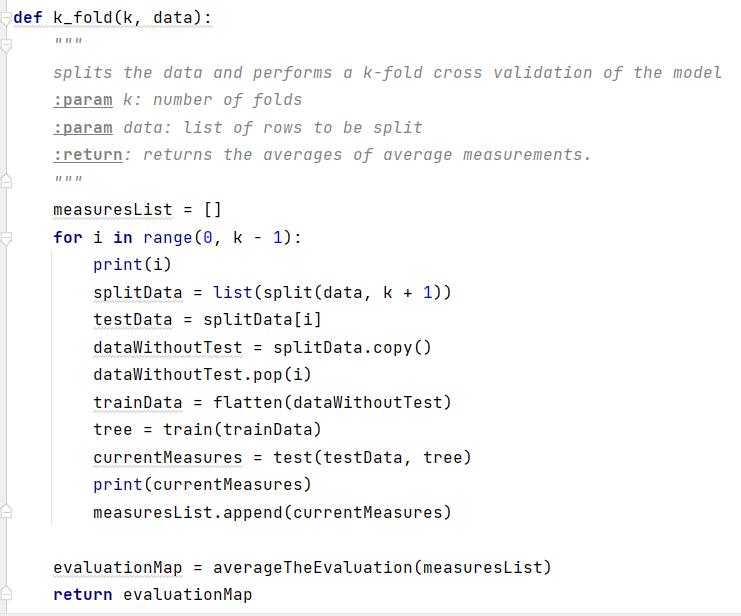
# Machine Learning – Lab 7 – Rad George-Rares – group 242

**Project 2, Component 4 & 5.**

**Experimental results and analysis**

The k-fold **cross validation** method was used with a k=10. The data was split into 10 folds, and for each iteration the model was trained with 90% folds and the one left out is used as testing data. After each iteration, another fold is used as testing.



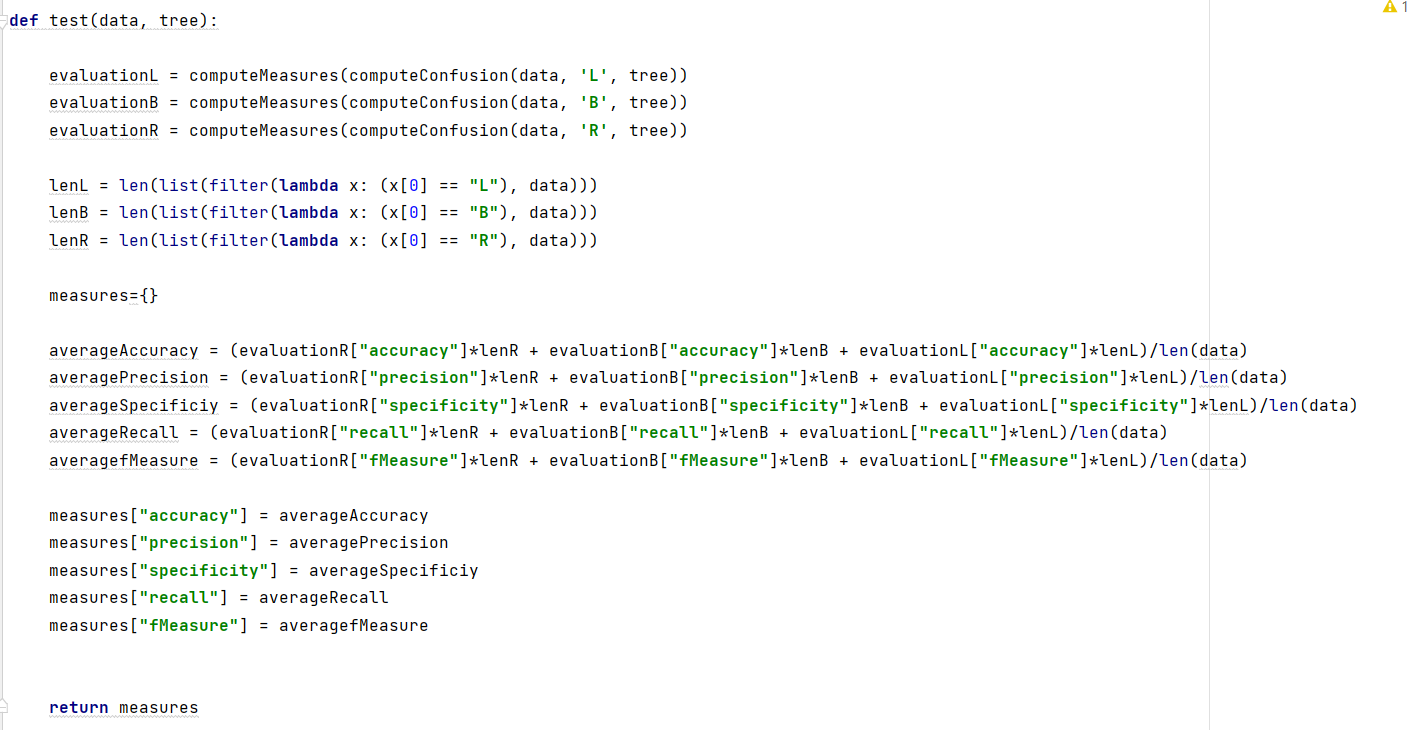
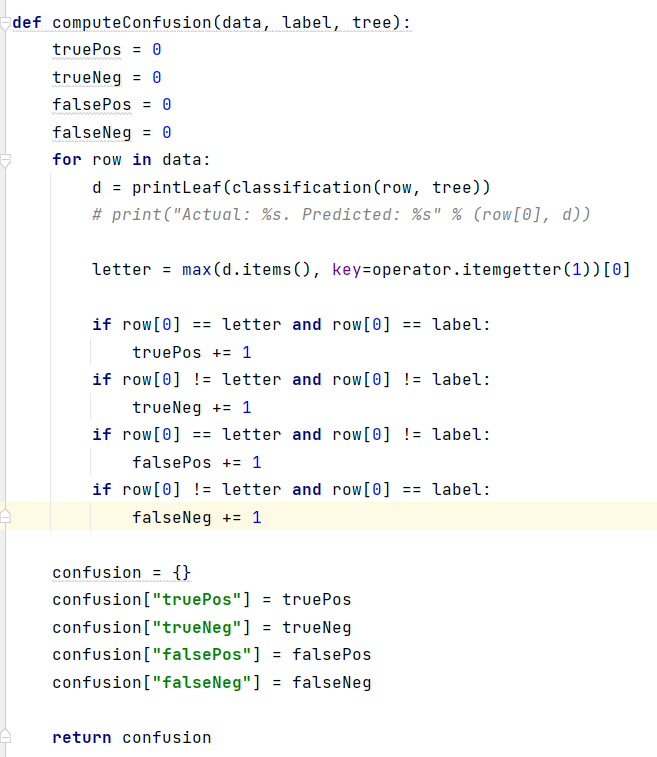
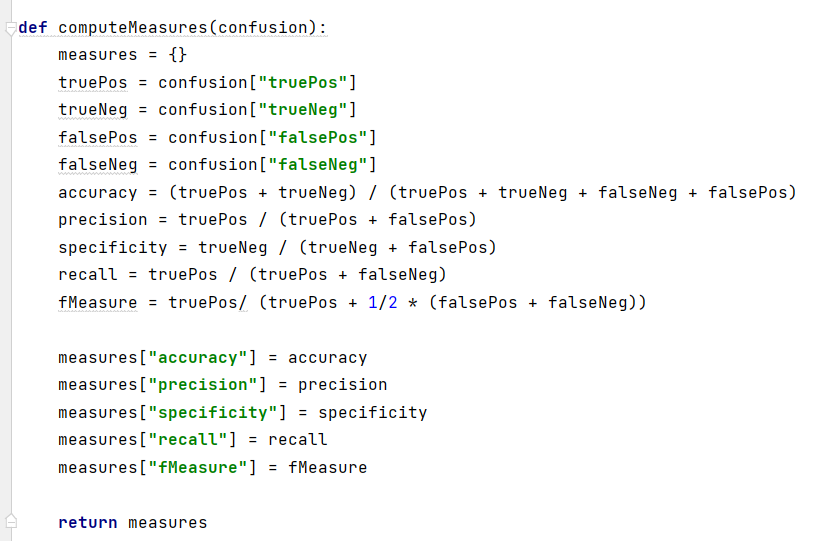
As for the **statistical analysis**, the 95% confidence level interval was yielded using the formula 1.96 \* sqrt( (accuracy \* (1 - accuracy)) / 50), and the macro accuracy of the combined three target classes, resulting in a interval

of +/- 13%.  
 In order to **evaluate** the model, I have used a method that computed the elements of the confusion matrix: the true positives, true negatives, false negatives and false positives that I further used in order to compute the accuracy, precision, recall, sensitivity and f-measure. These were computed in a global fashion(average of the results of each fold) after in each iteration of the cross validation, the weighted average of these evaluations of each class (L, B, R) was computed)

The initial results were done with a normal average and this produced really bad results (under 45% accuracy). After the switch to weighted average, the following results were found:

**accuracy: 0.55943; precision: 0.5636, specificity: 0.40131, recall: 0.68615, f- measure: 0.59711**

Another attempt was done for the improvement of these results by shuffling the data, since I observed that the data was not evenly split (i.e.: there were folds that had more Right balanced samples), but this only provided worse results with a lower value of about 4%.



**LATER EDIT:**

I have experimented with different values for the k number for the folds and got different results:

Best Results: k=60

{'accuracy': 0.65173, 'precision': 0.66748, 'specificity': 0.520827, 'recall': 0.7111, 'f-measure': 0.677392}

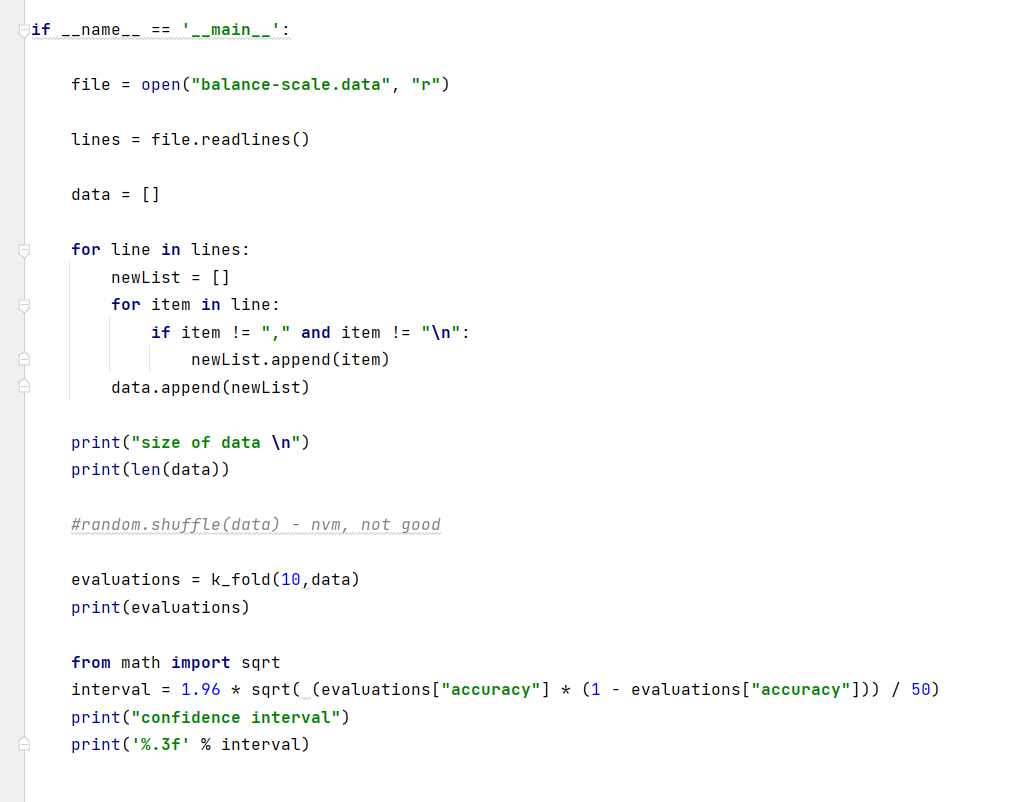
k=20

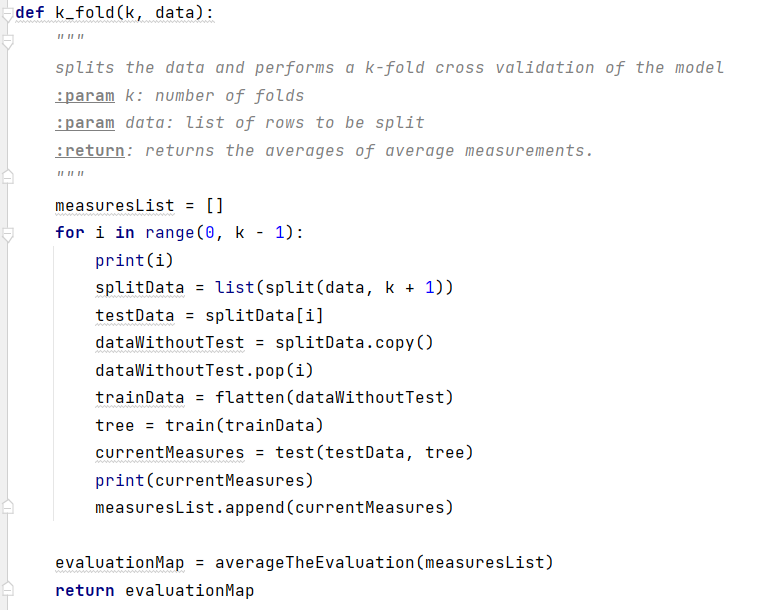
{'accuracy': 0.60283, 'precision': 0.601822, 'specificity': 0.46206, 'recall': 0.72008, 'f-measure': 0.638189}

k=100

{'accuracy': 0.64155, 'precision': 0.67509'specificity': 0.52686, 'recall': 0.68064, 'f-measure': 0.66726}

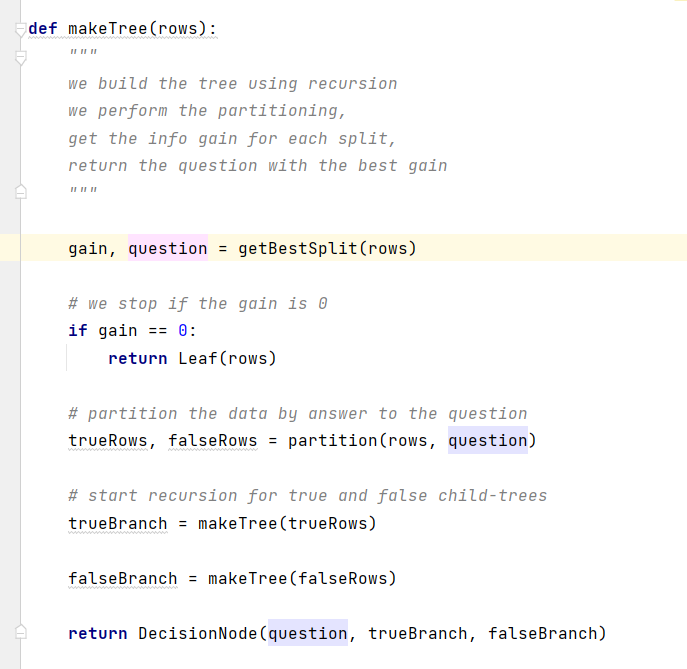
**User Manual:**

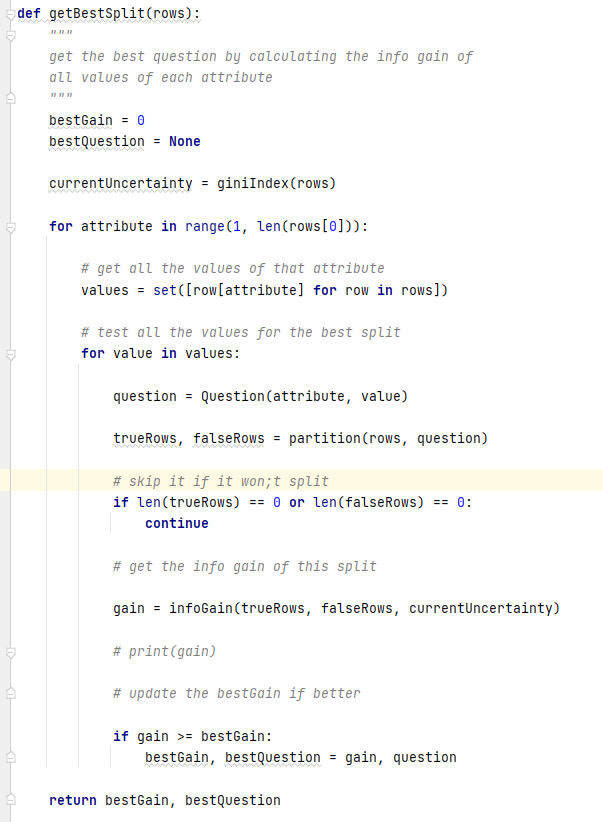
The process starts in the **main** method where the data is read and fed to the k\_fold method which returns the evaluation measures, after which they are printed. 

The **k\_folds** method splits the data in k folds, and applies the cross-validation for k times. The tree is created and trained with 90% of the data with the train method, tested with the 10% left in the test method and the evaluation measurements are returned from it and appended to the measurements container; this container is finally returned in the main method.

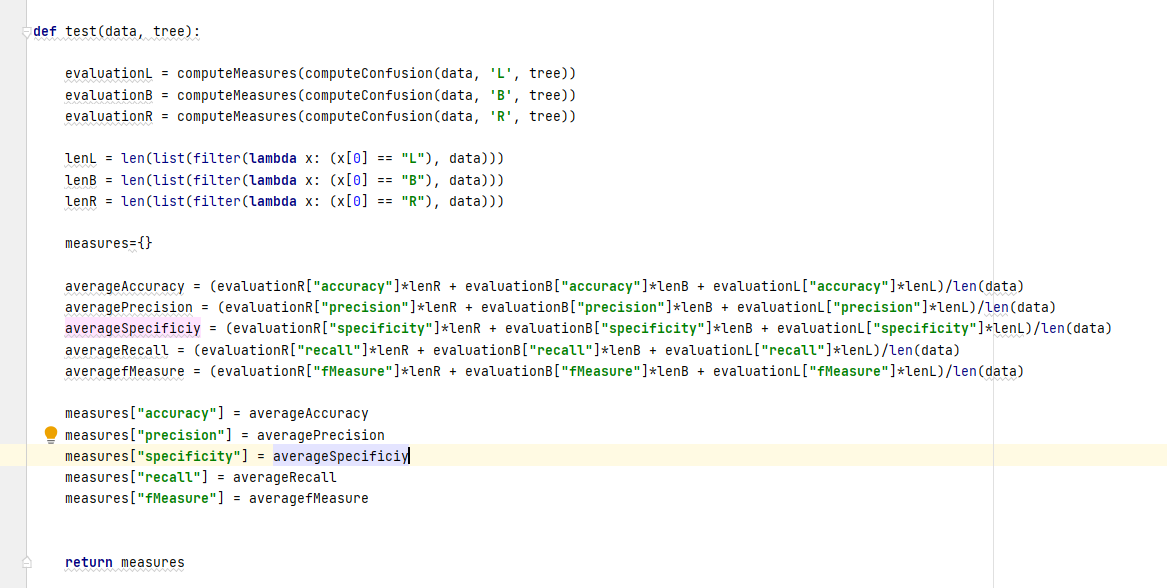
The **train** method is just a caller for the makeTree method which uses the data to create the tree.

This **makeTree** method gets the gain and the question by calling the getBestSplit method with the given data ; if the gain is 0 a Leaf object containing only the rows that were split by the previous DecisionNode returned. If not the rows are partitioned using the question by the partition method based on how to rows are “answering” the Question. After that two DecisionNodes are created by recursively calling this method again for each respective data, which contains the false and true DecisionNode with their respective Question. This method returns the root of the tree.

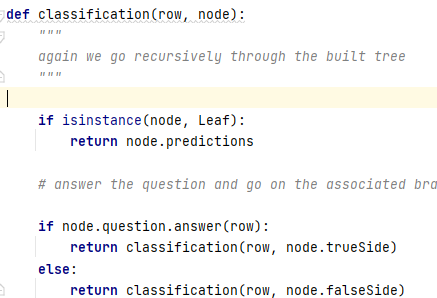


The **getBestSplit** method searches for all attributes of the rows, all the values of that attribute and creates a Question from the attribute and value. That Question is used to partition the dataset and it is updated as the best question if the partition actually split the dataset based on the Question and the information gain of that split is better than the last one; also the information gain is updated as the new best one in this case; At the end, the best gain and the best Question are split. 

The **test** method takes as input the data rows and the previously created tree. It computes the confusion matrices for each target class (L, B, R) and from that creates the evaluation measurements: accuracy, precision etc. These measurements are averaged with the weights of each target class and returned as a dictionary.



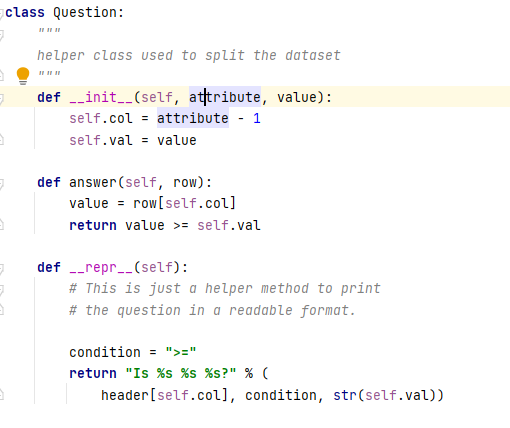
In the method computeConfusion the data is actually classified using the method **classification**, which takes as input a row and the root DecisionNode. It recursively goes through the Nodes and answers the Question inside each node. Then, it advances further on the next DecisionNode that met the condition of the Question. This stops when the current Node is a Leaf and the predictions in that Leaf are returned.



Some helper classes were defined that were mentioned above:

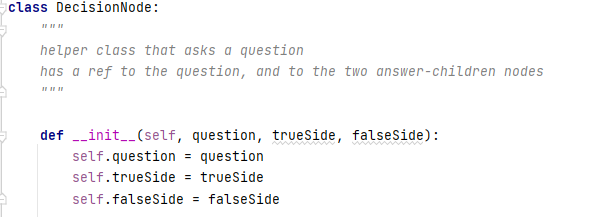
Question:

Holds the attribute that the Question is asked upon and the value of it.

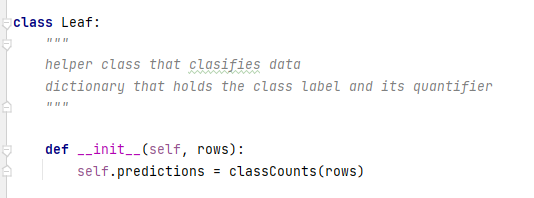


DecisionNode:

Holds a Question and two references to the false/true DecisionNodes.



Leaf:

Holds the predictions.

In order to run the project, one needs to run the main method of the main.py file. Or run the main.exe file directly.