Introduction to Statistical Computing in Scala - an Implementation of the K-Nearest Neighbors classifier

Roxana Tesileanu, *Research Assistant, National Institute of Forest Research and Management (INCDS), Romania, roxana.te@web.de*

***Abstract*—Statistical computing in ecology evolves at a high speed, mainly because researchers have recognized the advantage of being able to design their algorithms according to their needs. The present paper introduces the implementation in Scala of the k-Nearest Neighbors (kNN) classifier based on Euclidean distances, which can be applied also on small datasets, a situation commonly encountered in ecological research.**

***Index Terms*—machine learning in ecology, k-Nearest Neighbors classification, Scala language, Simple Build Tool (SBT), multivariate analyses**

I. Introduction

One of the drivers of the machine-learning progress is the great amount of data born within and gathered by networked and mobile computing systems which necessitates further processing in order to gain insights into the specific fields from which it originates (Jordan and Mitchell 2015, p. 256). The "Big Data phenomenon" is real but, unfortunately, it can not be generalized to all research areas, especially some areas of ecological research which investigate systems which are by nature data-poor and will probably remain as such unless cost-intensive data collection projects are being proposed and financed. This by no means implies that such areas cannot take advantage of the progress of machine-learning and take the best out of the existing datasets. On the contrary, machine-learning, being a study field which "sits at the crossroads of computer science, statistics and a variety of other disciplines concerned with automatic improvement over time, and inference and decision-making under uncertainty" (Jordan and Mitchell 2015, p. 256), is welcomed in real-world environmental decision-making facing the need of proposing courses of action under the cloud of uncertainties and regarding issues characterized by multiple attributes (Vatn and Bromley, 1994). Moreover, machine-learning is used not only in analyzing data from observational studies (which might benefit from larger time series datasets), but also from experimental studies (which can't produce large amounts of data but benefit from an experimental design, delivering data which provides stronger inference about effects and causal relationships - Quinn and Keough, 2002) (Harrington 2012). Indeed, a combination of machine-learning classifiers with the benefits of a MANOVA experimental design used for collecting training data, might produce useful research results, in ecological research too. Thus, in order to be able to use the power of machine-learning algorithms adapted to small ecological datasets, a statistical package written in Scala is currently under development at INCDS.

The idea of adapting the use of machine-learning algorithms to real-world data requirements is not new. Hastie and Tibshirani (1996), describe the Discriminant Adaptive Nearest Neighbor (DANN) procedure as a locally adaptive form of nearest neighbors classification using a modified linear discriminant analysis (LDA) procedure to estimate an effective metric for computing neighborhoods. The main idea of combining distance-based approaches with approaches based on the association between variables is driven by the fact that a multivariate dataset can be analysed in both ways (Quinn and Keough 2002): calculating scores for the derived variables (components) for each object, and calculating dissimilarity measures (distances) for every possible pair of objects.

The present paper introduces the Scala functions used to run the kNN algorithm and then it delineates the role of the classifier within a real-world classification pipeline for reducing error.

II. Running the kNN algorithm

In order to be able to run the kNN algorithm written using the Scala language, one should consider which tools to choose for the different components of a development system - which are according to Rehman and Paul (2003): the hardware platform, the operating system, editors, compilers and assemblers, debuggers, version control system and bug tracking. For implementing Scala projects, working with the Scala Build Tool (SBT) (http://www.scala-sbt.org/index.html) condenses the list of the components of a development system but also requires a default structure of the project directory with the source files of user-defined packages located in the src subdirectory (Suereth and Farwell 2016) following a standard pattern. This is the reason for which the source files of the scalaML package containing the kNN classifier are placed at the end of a chain of subdirectories within the project directory (see appendix A through E). Another advantage of using SBT for the statistical computing workflow, beside condensing the list of components of a development system, is the interactive use of the Scala REPL, a tool for evaluating expressions in Scala (https://docs.scala-lang.org/overviews/repl/overview.html), within a SBT session

(for further details and references see Tesileanu 2017 Linux-Scala.docx). In order to access the functions for the kNN classification, the package needs to be imported into the current REPL session.

The kNN classifier reads CSV and text files for classification tasks, uses basic vector operations for handling variables and objects, runs the kNN algorithm on the data and computes the error rate of the kNN classifier.

A. Basic vector operations

Because most statistical algorithms manipulate datasets which are actually collections of vectors, manipulating vectors is an essential task. The scalaML package offers this functionality, using the Scala main types while developing machine-learning algorithms. This makes the process of synchronizing types in further development stages easier. The vector operations are found in the BasicVectorOP.scala source file (see appendix A) and include: vector addition, vector subtraction, elementwise multiplication and dot product for any two vectors of type Array[Double], and matrix multiplication for any two matrices of type Array[Array[Double]], where each Array[Double] represents an observation (object), and all elements with the same index of inside arrays (i.e. index inside of any Array[Double] within the Array[Array[Double]]) represent a column (i.e. a variable) of the multivariate dataset (Dawkins 2005, Odersky et al. 2010, Swartz 2015, Trask 2017, Scala Standard Library 2.12.0. - EPFL 2003-2016).

The documentation of the kNN classifier offeres worked examples of functions' application in REPL (see Tesileanu 2017 INCDS Technical Report, https://www.researchgate.net/publication/318815119_Introduction_to_statistical_computing_in_Scala_an_application_of_the_k-Nearest_Neighbors_classifier).

B. Reading files for classification tasks

The ReadFile.scala source file (appendix B) of the scalaML package contains the function used to read CSV and text files for classification tasks. It returns a tuple with the information needed for the kNN algorithm (a data matrix of type Vector[Array[Double]], the data labels in the form of a Vector[Int], and the used classes in the form of a Range[Int]) (Harrington 2012, Odersky et al. 2010, Swartz 2015, Scala Standard Library 2.12.0. - EPFL 2003-2016) (see Tesileanu 2017 for details).

C. KNN classification function

The kNN classification function (see appendix C) implements in Scala the pseudocode of the kNN algorithm (Harrington 2012, Odersky et al. 2010, Swartz 2015, Scala Standard Library 2.12.0. - EPFL 2003-2016). It starts by calculating the Euclidean distance between a new object (which is going to be classified by the algorithm) and every object of the dataset, it sorts the distances in increasing order, it then takes k objects with the lowest distances and finds the majority class among these k objects, returning it as the prediction for the class of the new object (Harrington 2012). The documentation of the kNN classifier gives a thorough example of its use (see Tesileanu 2017). The variables can also be scaled from 0 to 1 before using the kNN classifier, if a normalization is required. For this purpose a normalizing function can be used (see appendix D) (Quinn and Keough 2002, Harrington 2012, Odersky et al. 2010, Scala Standard Library 2.12.0. - EPFL 2003-2016).

D. Testing framework

In order to test the accuracy of the kNN classifier, an error rate can be computed using the testing function provided in the TestFrameWorkClassif.scala source file (see appendix E) (Harrington 2012, Odersky et al. 2010, Scala Standard Library 2.12.0. - EPFL 2003-2016). The error rate is given by the total number of errors (misclassified observations) divided by the total number of tested observations (see Tesileanu 2017 for details on its use).

III. The learning pipeline for reducing error

The machine-learning workflow for classification tasks for high-dimensional datasets (i.e. more than three variables recorded for each object) generally consists of the following steps: data collection, reading the data in R, Scala, or other programming language used for statistical computation, analyzing the data using multivariate analyses like (MANOVA, PCA, cluster analysis, etc.), applying and testing the classification algorithm (kNN, Bayes classifier, etc.), deciding and argumenting the choice of the most appropriate model (Harrington 2012). Real-world workflows are itterative, requiring a dynamic configuration of the used algorithms and eventually the application of different algorithms to compare the results. Such workflows are called in machine-learning practice "learning pipelines" (Karau et al. 2015).

The present paper proposes a combination of the MANOVA experimental design and analysis, with a further data classification step using the kNN classifier. The Hintergrundidee is that, if two or more groups differ in their group centroids, they are, globally speaking, suitable for classification tasks. If, not, then we probably miss the variables which make a difference, or, we don't really have two different populations, and further classifications miss their point.

IV. Conclusion

Appendix A
Source file - src/main/scala/com/mai/scalaML/BasicVectorOP.scala

.....

Appendix B
Source file - src/main/scala/com/mai/scalaML/ReadFile.scala

.....

APPENDIX C
SOURCE FILE - SRC/MAIN/SCALA/COM/MAI/SCALAML/KNN.SCALA

.....

APPENDIX D
SOURCE FILE - SRC/MAIN/SCALA/COM/MAI/SCALAML/AUTONORM.SCALA

.....

APPENDIX E
SOURCE FILE - SRC/MAIN/SCALA/MAI/COM/SCALAML/TESTFRAMEWORKCLASSIF.SCALA

.....

ACKNOWLEDGMENT

The author would like to thank Jeff Druce and Mike Reposa from CRA for very useful learning tips and Georgeta Ionescu from INCDS for giving me the financial support needed. The present document was edited using the LaTeX document class IEEEtran developed by Michael Shell (Shell 2015).

NOTE

This paper is part of the project "Experimenting with Scala and R for multivariate analyses", which is stored on my GitHub profile under the following link: https://github.com/RoxanaTesileanu/multivariate_analyses.

Roxana Tesileanu graduated Forestry and Environmental Sciences (MSc.) in 2008 at Albert-Ludwigs Universitaet in Freiburg i. Br., Germany, and works since 2014 as research assistant at INCDS Romania. She is specialized in environmental statistics and bioinformatics.