Classifier Comparison: Code Base Manual

Code base version 1.1, last updated on March 7, 2018

Timo Deist

timo.deist@maastro.nl

Frank Dankers

frank.dankers@maastro.nl

MAASTRO clinic
Maastricht University Medical Center
Department of Radiation Oncology
www.maastro.nl

P.O. Box 3035
6202 NA Maastricht
The Netherlands

Dr. Tanslaan 12
6229 ET Maastricht
The Netherlands

Tel: 0031 88 44 55 666
Fax: 0031 88 44 55 667

Table of Contents

1. Introduction

The purpose of this manual is to explain the workings of the *R* code base accompanying the manuscript entitled *"Machine learning algorithms for outcome prediction in (chemo)radiotherapy: an empirical comparison of classifiers"* [1]. The manuscript provides advice for researchers on selecting machine learning algorithms to perform outcome prediction modelling in the field of radiotherapy. The conclusions drawn in the manuscript are based on numerical simulation of 6 classifiers (Appendix A) implemented in the popular *R* package *caret* [2], by Kuhn et al. 2016 [3], on a range of multiple datasets covering prediction of survival, and toxicities in lung and head and neck cancer.

The code base on the repository [4] may be helpful for fellow researchers in modelling treatment outcomes. The goal of the code base is to allow comparison of the average discriminating performance of the different classifiers for a given dataset. The code base is not meant for individual model creation.

If you have questions or remarks using the code base please contact one of the authors at the email-addresses listed on the title page of this manual.

If you use this code base in your research or a paper, please refer to the accompanying manuscript as follows [1].

All simulations using this code base have been tested on *RStudio* version 1.0.153 [6] and *R* version 3.4.1 [6].

2. Methodology

The experimental design is depicted in Figure 1. Each dataset is split into 5 stratified folds (step 1). For each of the folds, the data is pre-processed (imputation, dummy coding, deleting zero variance features, rescaling) (step 2). The tuned hyperparameters are determined in the training set via a 5-fold inner CV (steps 3-5). Based on the selected hyperparameters, a model is learned on the training set (step 6) and applied on the test set (step 7). Performance metrics are calculated on the test set (step 8) and stored for all folds. This process is repeated for all the classifiers. Randomization seeds are stable across classifiers within a repetition to allow pairwise comparison of classifier performance. The entire process is repeated 100 times.

The number of repetitions, outer folds and inner folds can be set through the main ini-file (see 5.4 Main ini-file). For additional information regarding the performance metrics, please see the manuscript [1].

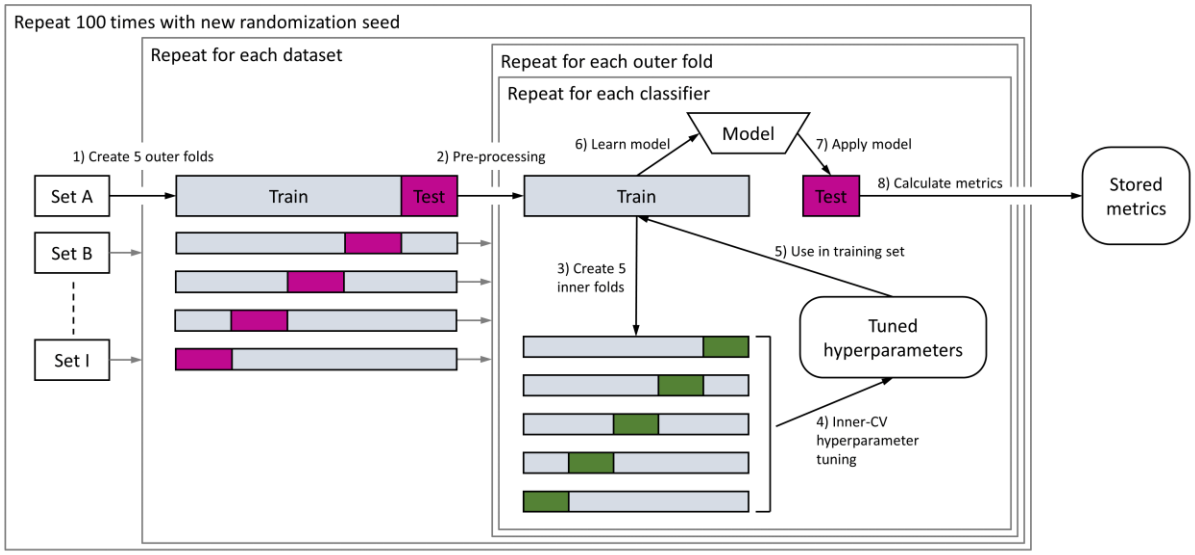

Figure 1: Schematic overview of the experimental design of the code base.

3. Directory structure

The code base requires the presence of three directories listed in the following table. The dot in the directory represents the installation/current directory. When you download the code base from the repository the "Output"-directory will be missing (due to Git-related technical reasons), so you need to create that directory manually.

Table 1: The required directories prior to running the code base are Code, Data and Output.

Directory	Explanation
".\Code\"	Contains the *R* code needed to run the simulations and output analysis.
".\Data\"	Contains the dataset(s) that you want to run through this code base. Each dataset should preferable be placed in a unique subdirectory. Note that datasets in this directory are not automatically included in the simulation run as this is determined by the main ini-file (see 5.4 Main ini-file). An alternative data directory can be set through the main ini-file.
".\Output\"	Will contain subdirectories of output generated by performing simulation runs. The simulation output is an RData-file and copies of the main ini- and csv-files that were used for the run. If you perform an output analysis run on the RData-file the analysis results are also stored in the subdirectory of the RData-file. An alternative output directory can be set through the main ini-file (see 5.4 Main ini-file).

4. Package installation

The code base requires the installation of several packages. This includes the installation of the *caret* package, version 6.0-73 [2-3].

Note: If you see deteriorated performance for classifier *svmRadial* please see Appendix F.

Additionally, there are packages required for several classifiers, for data imputation, plotting functionality etc. All required packages can be installed by running the script named "main_install_all_packages.R". This script needs to be run only once. A full list of required packages is given in Appendix B.

5. Perform simulation

The following sections (5.1 through 5.4) explain the necessary steps you need to undertake to be able to start a simulation run (5.5). If something is incorrectly set a simulation run is likely to encounter errors.

5.1. Datasets

Each dataset that you want to include in the simulation should consist of a single dataset csv-file and preferably be stored in a unique subdirectory in the ".\Data\" directory, e.g., ".\Data\Oberije_et_al_(2015)\ Stage3_anonymized.csv".

Within the dataset csv-file the patients should be given across rows, features and outcome(s) should be given across columns.

By default the Data-directory contains two publicly available datasets (www.cancerdata.org) that can be used to perform a test simulation run.

It is advisable to do any data processing via the dataset load function (see 5.3 Dataset loadfunction) and not directly on the dataset csv-file, so that a processing trail remains traceable.

5.2. Main csv-file

Before running the simulation on a dataset it is important to correctly set and verify parameters in the main csv-file. The name of the main csv-file is "main_simulation_datasetDefinitions.csv" and it is located in the ".\Code\" directory (enabling the display of file extensions in your operating system makes it easier to locate this file). It contains the required information about the dataset(s) that you can then include in a simulation run.

Note that not all datasets mentioned in the csv-file are automatically included in a simulation run as this is controlled by the main ini-file (see 5.4 Main ini-file).

You can edit the main csv-file through a text-editor or with Excel. The separator of the csv-file should be a comma ",". The repository version of this file contains example entries for publicly available datasets. The first line of the main csv-file always has to read:

datasetName;datasetFoldername;datasetFilename;datasetLoadfunction

Every dataset that you want to include should be entered on its own line. Each line should contain four dataset parameters explained to Table 2. An example of a dataset line in the main csv-file could be:

OberijeEtAl;Oberije_et_al_(2015);Stage3_anonymized.csv;loadDataOberijeEtAl

Table 2: Overview of entries to fill in inside the main csv-file and per dataset.

Parameter	Explanation
datasetName	Name of the dataset, used to select a dataset for a simulation run in the main ini-file **and** used in output analysis results (e.g., plots/tables etc.)
datasetFoldername	Name of the dataset subdirectory inside the data-directory ".\Data\"
datasetFilename	Name of the dataset csv-file (located in the dataset subdirectory) with .csv-extension
datasetLoadfunction	Name of the dataset load function (located in ".\Code\") without .R-extension (see 5.3 Dataset loadfunctions)

5.3. Dataset loadfunctions

The purpose of the load function is to load in the dataset (the dataset csv-file stored in its subdirectory in the data-directory) based on the information that is entered in the main dataset definitions csv-file (see 5.1 Datasets). Each dataset that you want to include in the simulation needs to have its own load function that is to be located in the ".\Code\" directory. You need to generate a load function for your dataset yourself.

The name of the load function for a given dataset should correspond with the name that you have set in the main csv-file (see 5.2 Main csv-file) **and** it should match with the function name inside the loadfunction R-file on the first line (e.g., "loadDataOberijeEtAl = function (pathFromDataFolderToCsv,pathToDataFolder)").

Example load functions for publicly available datasets are present in the code base on the repository (e.g., "loadDataOberijeEtAl.R").

The easiest way to get started is by copying an example load function and adjusting it to your needs. You will need to change the filename, change the load function functionname in the first line of its code and add the load function name to the dataset/line in the main csv-file (see 5.2 Main csv-file).

You can then best make sure your load function is working correctly by entering it using the *RStudio*'s debugging functionality. Set a breakpoint inside the load function, debugSource() the load function and run a simulation (continue with 5.4 Main ini-file). When you are inside your load function you can execute code line by line to evaluate its effects.

Make sure to examine the final result of your load function to check if the dataset loading has been performed correctly, i.e. look at the load function dataframe output using *RStudio*'s debugging functionality.

Some help on data processing steps you might need is provided below.

5.3.1. Read the dataset csv-file

At the start of the load function it should read the dataset csv-file and define the separator and decimal characters as well as definition(s) for missing values.

```
# read csv file
pathToFile = file.path(pathToDataFolder,pathFromDataFolderToCsv)
data = read.csv(pathToFile,sep = ";", dec = ".", strip.white=TRUE, na.strings =
c("UNK","NA","na"))
```

5.3.2. Remove unwanted columns

Columns to remove can be indicated via standard *R* code for data frames (e.g., patient ID's or outcomes you don't want to model (prevent data leakage!)).

```
# columns to drop, feature/outcome
drop = c('ID','localcontrol,'5yearsurvival')
data = data[ , !(names(data) %in% drop)]
```

5.3.3. Keep wanted columns

Alternatively, you can define the columns to keep.

```
# columns to keep
keep = c('Sex', 'Age', 'T_stage', 'meandose')
data = data[, keep]
```

5.3.4. Forcing data types

While inside the debug-mode you can expand the data frame in the upper right workspace to see if the feature types (integers, logicals, numericals or factors (i.e. categoricals)) were correctly assigned by the read csv function. If necessary, manually convert features to factors or numericals (make sure the outcome feature is a factor).

Note: if forcing into numericals is necessary in your case this might be indicative of a faulty decimal indicator in the earlier *read.csv* function call.

```
# convert to numericals
toNumericals = c('PTVmindose','PTVmaxdose')
data[,toNumericals] <- sapply(data[, toNumericals], as.numeric)

# convert to factors
toFactors = c('Arm','zhs','siteprimary','histology')
data[toFactors] = lapply(data[toFactors], factor)
```

5.3.5. Separating the outcome

The output parameter should be extracted and assigned to a separate variable (*data_class* in the example below) and removed from the data frame containing the input features. If the output is not yet in binary form, then it should be dichotomized. It should also be a factor for the revalue-function (that turns it into event/nonEvent) to work.

```
# outcome variable
```

```
data_class = data$survival # place outcome in separate variable
data$survival = NULL      # remove outcome from data
data_class = revalue(as.factor(data_class), c('0' = 'nonEvent', '1' = 'event')) # label as
(non)event
```

5.4. Main ini-file

In the main ini-file, "main_simulation_parameters.ini" located in the ".\Code\" directory, the simulation parameters are set (enabling the display of file extensions makes it easier to locate this file).

Firstly, it is necessary to correctly set the full paths for the ".\Data\" and ".\Output\" directories using the parameters *pathToDataFolder* and *pathToOutputFolder* (these paths allow backward slashes).

The parameters include the minimum and maximum repetition number (this allows distribution repetitions over multiple simulation runs to speed up the run by parallelization, see Appendix G), the number of outer and inner folds, the classifiers to use and the datasets to include in the simulation (they must be listed in the main csv file, see 5.2 Main csv-file).

Additionally, you can anonymize the output data (you can also anonymize at the output analysis stage, see 6 Output analysis), set the randomization seed and turn saving of output on or off. The directories for the datasets and the output can be set, and the main csv-filename is defined in the ini-file. If you enter backslashes in the paths for the dataset and output directories (Windows convention) these will be automatically converted to forward slashes (*Unix* convention).

Note that the last line of the ini-file should be empty to prevent warnings when the ini-file is read at the beginning of the simulation run.

Finally, you can turn default tuning of classifier hyperparameters on or off. When using default tuning *caret* tunes the hyperparameters listed in Appendix A using 3 default values per parameter. For instance, if the classifier contains two hyperparameters that will be tuned this will lead to a 3x3 grid (9 values) of hyperparameters. More extensive tuning can be performed by setting the default tuning to FALSE, which will result in random tuning with tunelength 25 (this number is hardcoded in the fit[Classifier].R functions and can be adjusted there). Twenty-five random hyperparameter values, or combinations of hyperparameter values if the classifier contains more than one hyperparameter) are evaluated. In the accompanying manuscript [1] all simulations are performed using random tuning. Some custom tuning parameters are set for certain classifiers via their respective fit-functions (listed in Appendix A). If you want to tune other hyperparameters than those listed in Appendix A you will need to write your own optimization code.

5.5. Start the run

When all of the information given in paragraphs 5.1 through 5.4 has been considered it is time to start a simulation by running "main_simulation.R". At the top of the script, it is **imperative** to correctly point the "*this.dir =* " setting on line 9 to the absolute path of the Code directory. Make sure to use **forward** slashes in the path. You only have to make this edit once.

During the simulation run progress information will be displayed in *RStudio*'s console window during the run. It will first display the parameters that were read from the main ini-file. Then it will show the results

of loading the dataset(s), reporting the number of features, patients and prevalence. Then the actual simulation will start by looping in order of repetitions, datasets, outer folds and classifiers.

A timer indicating the approximate time left to complete the simulation is displayed at the start of each repetition (from repetition 2 onwards). The runtime of this simulation is dependent on the number of datasets, repetitions and classifiers (approximately 15 seconds per dataset per repetition per classifier per outer fold). For the manuscript, this took approximately 6 days on a Intel Core i5-6200U CPU (12 datasets, 100 repetitions and 6 classifiers).

Simulation results will be saved in a RData-file in a subdirectory of the ".\Output\" directory when the run is finished. The naming convention of the subdirectory is: "[date]_[time]_[computer name] _[number of dataset]datasets_[number of classifiers]classifiers _reps[minRep-maxRep]". After a run it should contain an RData-file with the simulation results. This file is the input of the output analysis script (see 6 Output analysis). Additionally, copies of the main-csv and main-ini files are stored in the this Output subdirectory as well, so that the initialization parameters of the run can always be deduced afterwards.

The generated models are also saved during each repetition per dataset in another Output subdirectory named: "[date]_[time]_[computer name] _[number of dataset]datasets_[number of classifiers]classifiers _models". These models are **not** used for any subsequent analysis. This folder can take up significant disk space, but can be safely removed after the simulation run is completed if desired.

Additionally, after a repetition is finished the accumulated outputTable (which is the input necessary for the output analysis, see 6 Output analysis) is saved as a csv-file in another Output subdirectory named: "[date]_[time]_[computer name] _[number of dataset]datasets_[number of classifiers]classifiers _outputTables". If the simulation crashes you will at least have the outputTable up until the crash.

6. Output analysis

Once a simulation has finished the results can be analyzed through the script "main_output_analysis.R". Again, it is **imperative** to first correctly point the *"this.dir ="* setting to the absolute path of the Code directory (you only need to do this once). Use **forward** slashes in the path.

Prior to performing an output analysis it is possible to set a number of options through an ini-file named "main_output_analysis.ini". It allows the setting of saving the analysis output, excluding datasets, anonymization and classifier/dataset ordering.

Additionally, you can set the desired classifier labels to be used in the figures. This vector of strings should have the same length as the number of classifiers that are present. Additionally, the order should match 'prefClassifierOrder' ("main_output_analysis.ini"), or if it is empty the 'classifierSelection' order ("main_simulation_parameters.ini").

Note that changing classifier or dataset order can lead to slightly different numbers for AUC *rank* and AUC *rank* derived metrics, since tied AUCs are ranked randomly (to not benefit classifiers based on the specified order).

Running the main output analysis script will pop-up a file selection dialog window asking for the RData-file with the simulation result (located in a subdirectory of the Output directory). If you did a parallelized

run you can also select multiple RDATA-files (see Appendix G). Selecting the RDATA file(s) will continue the evaluation via a multitude of subfunctions (Appendix D). These functions will perform aggregation of folds, datasets and repetitions, and calculate a variety of performance metrics.

The runtime of this analysis is mainly dependent on the number of datasets, repetitions and classifiers (approximately 0.1 second per dataset per repetition per classifier). For the manuscript, this took less than 10 minutes (12 datasets, 100 repetitions, 6 classifiers).

The results of the output analysis are presented in tables and figures, which are saved in the same output subdirectory as the RData-file that has been analyzed. Figures include a combined AUC *rank* box and scatterplot of classifiers, heatmaps of AUC and AUC *rank* performance of classifiers per dataset and pairwise comparisons of classifiers. A table is generated with classifier performance by metric, and there are tables related to the classifier selection simulation experiment as described in the manuscript [1].

For a more comprehensive discussion regarding the analysis of the simulation results please refer to the manuscript.

References

1. T.M. Deist, F.J.W.M. Dankers et al., "Machine learning algorithms for outcome prediction in (chemo)radiotherapy: an empirical comparison of classifiers", [journal], vol. x, no. x, pp. xxx-xxx, xxx. 2017.
2. Link to *R* package *caret*: http://topepo.github.io/caret/index.html
3. M. Kuhn et al., "caret: Classification and Regression Training," 2016.
4. Link to code base repository: https://github.com/timodeist/classifier_selection_code
5. Link to R: https://cran.r-project.org/bin/windows/base/
6. Link to RStudio: https://www.rstudio.com/products/rstudio/download/
7. J. Friedman, T. Hastie, and R. Tibshirani, "Regularization Paths for Generalized Linear Models via Coordinate Descent," J. Stat. Softw., vol. 33, no. 1, pp. 1–22, 2010.
8. A. Liaw and M. Wiener, "Classification and Regression by randomForest," *R* News, vol. 2, no. 3, pp. 18–22, 2002.
9. W. N. Venables and B. D. Ripley, Modern Applied Statistics with S, Fourth. New York: Springer, 2002.
10. A. Karatzoglou, A. Smola, K. Hornik, and A. Zeileis, "kernlab – An S4 Package for Kernel Methods in R," J. Stat. Softw., vol. 11, no. 9, pp. 1–20, 2004.
11. J. Tuszynski, caTools: Tools: moving window statistics, GIF, Base64, ROC AUC, etc. 2014.
12. T. Therneau, B. Atkinson, and B. Ripley, rpart: Recursive Partitioning and Regression Trees. 2017.

Appendix A Classifiers

Table 3: An overview of classifiers that are implemented in the code base. All classifiers have built-in feature selection.

Classifier	*caret* label	*R* package	Dummy coding	Hyper-parameters
Logistic elastic net regression	*glmnet*	*glmnet* [7]	Yes	α, λ
Random forest	*rf*	*randomForest* [8]	No	$mtry$
Single-hidden-layer neural network	*nnet*	*nnet* [9]	No	$size, decay$
Support vector machine with radial basis function kernel	*svmRadial*	*kernlab* [10]	Yes	σ, C
LogitBoost	*LogitBoost*	*caTools* [11]	Yes	$nIter$
Decision tree	*rpart*	*rpart* [12]	No	cp

Appendix B Required *R* packages

Table 4: A list of required R packages prior to running the code base. These packages can be installed by running the script named "main_install_all_packages.R".

Name	Purpose
caret	Functions for streamlining different steps of predictive modelling
caTools	Classifier LogitBoost
ggplot2	Generating plots, e.g., boxplots, heatmaps
glmnet	Classifier generalized logistic elastic net regression
imputeMissing	Functions for handling of missing values
kernlab	Classifier SVM (support vector machine)
klaR	Model classification and visualization
nnet	Classifier neural network
plyr	Functions for variable manipulation, e.g., revalue()
randomForest	Classifier random forest
resourceSelection	Functions for Hosmer-Lemeshow test
rpart	Classifier rpart (decision tree)
svDialogs	Functions for user interface dialog windows (e.g., file selection)

Appendix C Main scripts

Table 5: An overview of the scripts and their purpose.

Names	Purpose
main_install_all_packages.R	Main script for package installation (see 4 Package installation).
main_simulation.R	Main script for performing a simulation (see 5.5 Start the run).
main_output_analysis.R	Main script for simulation output analysis (see 6 Output analysis).

Appendix D Functions

Table 6: An overview of the functions, grouped by purpose.

Name	Purpose
loadData[dataset name]	Custom functions per dataset to perform loading of the dataset csv-files (see 5.3 Dataset loadfunction).

runCvForClassifiers.R preprocess_dataset.R preprocess_imputeDataset.R preprocess_removeZeroVarianceColumns.R	Perform outer-fold cross-validation and the dataset pre-processing before simulation run (see 2 Methodology).
runClassifier.R fit[classifier name].R	Performs the fit of current training/test slices for a classifier using *caret* functionality (includes the inner folds required for hyperparameter tuning).
generate[x].R aggregate[x].R add[x].R	Functions used for analysis of simulation run output. These functions add metrics, aggregate over folds and repetitions and generate data frames for plotting or saving of tables.

Appendix E Code base version management

Table 7: An overview of version history of the repositories' code base.

Version	Date	Change
1.0	November 16, 2017	Initial code base version and repository manual
1.1	March 7, 2018	Updated code base version and repository manual after peer review

Appendix F Fix for poor *svmRadial* performance (*caret* version downgrade)

The manuscript results were generated employing various open-source *R* packages interfaced with the *R* package *caret*, version 6.0-73 (Kuhn et al. 2016 [3]). Deteriorated performance (i.e. low AUC values) for classifier *svmRadial* was seen using *caret*, 6.0-77. If you notice this behavior you can downgrade *caret* to the older version with the following steps:

First, install the Rtools package that matches your *R* version:

https://cran.r-project.org/bin/windows/Rtools/

Now remove and reinstall *caret* using these commands in the *RStudio* console window:

```
remove.packages("caret")
install.packages("devtools")
require(devtools)
install_version("caret", version = "6.0-73", repos = "http://cran.us.r-project.org")
```

Check the installed *caret* package version with these commands in the *RStudio* console window:

```
ip <- as.data.frame(installed.packages()[,c(1,3:4)])
rownames(ip) <- NULL
ip <- ip[is.na(ip$Priority),1:2,drop=FALSE]
```

```
print(ip, row.names=FALSE)
```

Appendix G Implemented fix for LogitBoost

The code base contains an implemented fix for the classifier LogitBoost. The LogitBoost predict function that is used to generate classified predictions from the predicted probabilities gives NAs whenever there is a 0.5 predicted probability for a positive outcome (and thus also for a negative outcome). Patient rows containing these NAs are omitted in the subsequent Accuracy and Cohen's Kappa calculation (but not for AUC calculation). This leads to too optimistic values for the performance metrics Accuracy and Cohen's Kappa. The implemented fix in the code base overwrites these NAs with positive outcomes (simulating rounding of 0.5 upwards), so that these rows are not omitted anymore, leading to correct performance metrics.

Appendix H Reduce simulation runtimes by parallelization

Running the code base on many datasets using a large number of repetitions can lead to long runtimes (for the accompanying manuscript the runtime was 6 days for 12 datasets, 6 classifiers and 100 repetitions). The code base allows basic parallelization, e.g., over CPU cores or over additional systems, by running multiple *RStudio* instances and starting the simulation several times with different minimum and maximum repetition numbers in the ini-file. Make sure to update and save the "main_simulation_parameters.ini" file correctly before you start the simulations using "main_simulation.R" (e.g., for simulation one use minRep=1 and maxRep=10, for simulation two use minRep=11 and maxRep = 20, etc.).

It is advisable to not start *RStudio* instances than you have CPU cores available (or even one less than the number of cores so that the system remains responsive enough).

After the simulations across multiple *RStudio* instances are finished you need to combine the resulting RDATA-files in a single folder. You can then run an output analysis (using "main_output_analysis.R") and select the multiple RDATA files. A single output analysis will be generated.