# dtComb: A Comprehensive R Package for Combining Two Diagnostic Tests

S. İlayda YERLİTAŞ TAŞTAN[1,2], Serra Berşan GENGEÇ[2], Necla KOÇHAN[3], Gözde ERTÜRK ZARARSIZ[1,2], Selçuk KORKMAZ[4] and Gökmen ZARARSIZ[1,2†]

[1] *Erciyes University Faculty of Medicine Department of Biostatistics, Kayseri, Türkiye*

[2] *Erciyes University Drug Application and Research Center (ERFARMA) , Kayseri, Türkiye*

[3] *Izmir University of Economics-Department of Mathematics, 35330, İzmir, Türkiye*

[4] *Trakya University Faculty of Medicine Department of Biostatistics, Edirne, Türkiye*

October 10, 2024

**Abstract**

`dtComb` is a comprehensive R package that combines two different diagnostic tests. Using its extensive collection of 143 combination methods, the `dtComb` package enables researchers to standardize their data and merge diagnostic tests. Users can load the dataset containing the reference list and the diagnostic tests they intend to utilize. The package includes combination methods grouped into four main categories: linear combination methods (`linComb`), non-linear combination methods (`nonlinComb`), mathematical operators (`mathComb`), and machine-learning algorithms (`mlComb`). The package incorporates eight specific combination methods from the literature within the scope of linear combination methods. Non-linear combination methods encompass statistical approaches like polynomial regression, penalized regression methods, and splines, incorporating the interactions between the diagnostic tests. Mathematical operators involve arithmetic operations and eight `distance measures` adaptable to various data structures. Finally, machine-learning algorithms include 113 models from the `caret` package tailored for `dtComb`'s data structure. The data standardization step includes five different methods: Z-score, T-score, Mean, Deviance, and Range standardization. The `dtComb` integrates machine-learning approaches, enabling the utilization of preprocessing methods available in the `caret` package for standardization purposes within the `dtComb` environment. The `dtComb` package allows users to fine-tune hyperparameters while building a model. This is accomplished through resampling techniques such as 10-fold cross-validation, bootstrapping, and 10-fold repeated cross-validation. Since machine-learning algorithms are directly adapted from the `caret` package, all resampling methods available in the `caret` package are applicable within the `dtComb` environment. Following the model building, the `predict` function predicts the class labels and returns the combination scores of new observations from the test set. The `dtComb` package is designed to be user-friendly and easy to use and is currently the most comprehensive package developed to combine diagnostic tests in the literature. This vignette was created to guide researchers on how to use this package. **dtComb version:** 1.0.4

## 1 Introduction

Diagnostic tests are critical in distinguishing diseases and determining accurate diagnoses for patients, and they significantly impact clinical decisions. Beyond their fundamental role in medical diagnosis, these tests also aid in developing appropriate treatment strategies while lowering treatment costs. The widespread availability of these diagnostic tests depends on their accuracy, performance, and reliability. When it comes to diagnosing medical conditions, there may be several tests available, and some may perform better than others and eventually replace established protocols. Studies have shown that using multiple tests rather than relying on a single test improves diagnostic performance [1, 2, 3]. A number

of approaches to combining diagnostic tests are available in the literature. The *dtComb* package includes a variety of combination methods existing in the literature, data standardization approaches for different data structure, and resampling methods for model building. In this vignette, users will learn how to combine two diagnostic tests with different combination methods. *dtComb* package can be loaded as below:

```
library(dtComb)
```

## 2 Preparing the input data

The methods provided within this package are designed to require a `DataFrame` comprising three columns, where the first column represents class labels, and the subsequent columns correspond to the values of the corresponding markers. The class label is a binary variable (i.e., negative/positive, present/absent) representing the outcomes of a reference test used in disease precision. This vignette will use the dataset `exampleData1`, included in this package. This dataset contains information from patients admitted to the General Surgery Department of Erciyes University Medical Faculty with complaints regarding abdominal pain. The dataset comprises 225 patients split into two groups: those requiring immediate laparotomy (110 patients) and those not requiring it (115 patients). Patients who had surgery due to postoperative pathologies are in the first group, whereas those with negative laparotomy results belong to the second group [4].

```
data(exampleData1)
head(exampleData1)

##    group ddimer log_leukocyte
## 1 needed   8.09          5.52
## 2 needed   5.16          4.43
## 3 needed   8.90          5.20
## 4 needed  10.17          5.39
## 5 needed   1.93          5.09
## 6 needed   3.63          4.68
```

The dataset is divided into two parts: the training and the test sets. The training set consists of 75% of the dataset and is used to train classification models and to compare different model performances. The remaining portion of the dataset is saved as the `test` set, which will later be used in the prediction phase. The `train` and the `test` sets are built as follows:

```
# # train set from the exampleData1
set.seed(2128)
inTrain <- caret::createDataPartition(exampleData1$group, p = 3 / 4, list = FALSE)
trainData <- exampleData1[inTrain, ]
head(trainData)

##    group ddimer log_leukocyte
## 2 needed   5.16          4.43
## 3 needed   8.90          5.20
## 4 needed  10.17          5.39
## 5 needed   1.93          5.09
## 6 needed   3.63          4.68
## 7 needed   3.12          5.20
```

```
# # test set from the exampleData1
set.seed(2128)
testData <- exampleData1[-inTrain, -1]
```

We have a total of 170 patients in the training set, with 83 requiring laparotomy and 87 not requiring laparotomy. The training dataset is divided into two parts: `markers` (i.e., diagnostic test results) and `status` (i.e., reference test results or class labels). The class label is also converted into a factor variable if it is not a factor. The remaining 55 patients are assigned to the test set.

```
markers <- trainData[, -1]
status <- factor(trainData$group, levels = c("not_needed", "needed"))
```

# 3 Available methods

The *dtComb* package contains 143 methods for combining diagnostic tests. These methods are classified as linear methods, non-linear methods, mathematical operators, and machine-learning (ML) algorithms, each briefly explained below.

**Notations:**

Before getting into these methods, let us introduce some notations used throughout this vignette. Let $D_i$, $i = 1, 2, \ldots, n_1$ be the marker values of $i$th individual in diseased group, where $D_i = (D_{i1}, D_{i2})$, and $H_j$, $j = 1, 2, \ldots, n_2$ be the marker values of $j$th individual in healthy group, where $H_j = H_{j1}, H_{j2}$. Let $x_{i1} = c(D_{i1}, H_{j1})$ be the values of the first marker, and $x_{i2} = c(D_{i2}, H_{j2})$ be values of the second marker for the $i$th individual $i = 1, 2, ..., n$. Let $D_{i,min} = min(D_{i1}, D_{i2})$, $D_{i,max} = max(D_{i1}, D_{i2})$, $H_{j,min} = min(H_{j1}, H_{j2})$, $H_{j,max} = max(H_{j1}, H_{j2})$ and $c_i$ be the resulting combination score for the $i$th individual.

## 3.1 Linear combination methods:

- **Logistic Regression** (`logistic`): Combination score obtained by fitting a logistic regression model is as follows:

$$c_i = \left( \frac{e^{\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2}}}{1 + e^{\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2}}} \right)$$

  A combination score obtained by fitting a logistic regression model typically refers to the predicted probability or score assigned to each observation in a dataset based on the logistic regression model's fitted values [5].

- **Scoring based on Logistic Regression** (`scoring`): The combination score is obtained using the slope values of the relevant logistic regression model, slope values are rounded to the number of digits taken from the user [6].

$$c_i = \beta_1 x_{i1} + \beta_2 x_{i2}$$

- **Pepe & Thompson's method** (`PT`): The Pepe and Thompson combination score, developed using their optimal linear combination technique, aims to maximize the Mann-Whitney U statistic like the Min-max method. Unlike the Min-max method, the Pepe and Thomson method considers all marker values instead of the lowest and maximum values [7].

$$maximize\, U(\alpha) = \left( \frac{1}{n_1, n_2} \right) \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} I(D_{i1} + \alpha D_{i2} >= H_{j1} + \alpha H_{j2})$$

$$c_i = x_{i1} + \alpha x_{i2}$$

- **Pepe, Cai & Langton's method** (`PCL`): Pepe, Cai and Langton combination score obtained by using AUC as the parameter of a logistic regression model [8].

$$maximize\, U(\alpha) = \left( \frac{1}{n_1, n_2} \right) \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} I(D_{i1} + \alpha D_{i2} > H_{j1} + \alpha H_{j2}) + \left( \frac{1}{2} \right) I(D_{i1} + \alpha D_{i2} = H_{j1} + \alpha H_{j2})$$

$$c_i = x_{i1} + \alpha x_{i2}$$

- **Min-Max method (`minmax`)**: This method linearly combines the minimum and maximum values of the markers by finding a parameter, $\alpha$ , that maximizes the Mann-Whitney statistic, an empirical estimate of the ROC area [9].

$$maximize\ U(\alpha) = \left(\frac{1}{n_1, n_2}\right) \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} I(D_{i,max} + \alpha D_{i,min} > H_{j,max} + \alpha H_{j,min})$$

$$c_i = x_{i,max} + \alpha x_{i,min}$$

where $x_{,max} = max(x_{i1}, x_{i2})$ and $x_{,min} = min(x_{i1}, x_{i2})$.

- **Su & Liu's method (`SL`)**: The Su and Liu combination score is computed through Fisher's discriminant coefficients, which assumes that the underlying data follow a multivariate normal distribution, and the covariance matrices across different classes are assumed to be proportional [10]. Assuming that $D \sim N(\mu_D, \sum_D)$ and $H \sim N(\mu_H, \sum_H)$ represent the multivariate normal distributions for the diseased and non-diseased groups, respectively. The Fisher's coefficients are as follows:

$$(\alpha, \beta) = (\sum_D + \sum_H)^{-1} \mu$$

where $\mu_{=}\mu_D - \mu_H$. The combination score in this case is:

$$c_i = \alpha x_{i1} + \beta x_{i2}$$

- **Minimax approach (`minimax`):** Combination score obtained with the Minimax procedure; $t$ parameter is chosen as the value that gives the maximum AUC from the combination score [11]. Suppose that D follows a multivariate normal distribution $D \sim N(\mu_D, \sum_D)$, representing the diseased group, and H follows a multivariate normal distribution $H \sim N(\mu_H, \sum_H)$, representing the non-diseased group. Then Fisher's coefficients are as follows:

$$(\alpha, \beta) = [t\sum_D + (1-t)\sum_H]^{-1}(\mu_D - \mu_H)$$

$$c_i = b_1 x_1 + b_2 x_2$$

- **Todor & Saplacan's method (`TS`)**: Combination score obtained using the trigonometric functions of the $\Theta$ value that optimizes the corresponding AUC [12].

$$c_i = sin(\theta) x_{i1} + cos(\theta) x_{i2}$$

## 3.2 Nonlinear combination methods:

- **Logistic Regression with Polynomial Feature Space (`polyreg`)**: The method builds a logistic regression model with the polynomial feature space and returns the probability of a positive event for each observation.

- **Ridge Regression with Polynomial Feature Space (`ridgereg`)**: Ridge regression is a shrinkage method used to estimate the coefficients of highly correlated variables and in this case the polynomial feature space created from two markers. For the implementation of the method, the `glmnet` library is used with two functions: `cv.glmnet` to run a cross-validation model to determine the tuning parameter $\lambda$ and `glmnet` to fit the model with the selected tuning parameter [13]. For Ridge regression, the `glmnet` package is integrated into the dtComb package to facilitate the implementation of this method.

- **Lasso Regression with Polynomial Feature Space (`lassoreg`)**: Lasso regression, like Ridge regression, is a type of shrinkage method. However, a notable difference is that Lasso tends to set some feature coefficients to zero, making it useful for feature elimination. It also employs cross-validation for parameter selection and model fitting using the `glmnet` library [13].

- **Elastic-Net Regression with Polynomial Feature Space (`elasticreg`)**: Elastic-Net Regression is a hybrid model that merges the penalties from Ridge and Lasso regression, aiming to leverage the strengths of both approaches. This model involves two parameters: $\lambda$, similar to Ridge and Lasso, and $\alpha$, a user-defined mixing parameter ranging between 0 (representing Ridge) and 1 (representing Lasso). The $\alpha$ parameter determines the balance or weights between the loss functions of Ridge and Lasso regressions [13].

- **Splines (`splines`)**: Another non-linear approach to combining markers involves employing regression models within a polynomial feature space. This approach applies multiple regression models to the dataset using a function derived from piecewise polynomials. This implementation uses splines with user-defined degrees of freedom and degrees for the fitted polynomials. The `splines` library is employed to construct piecewise logistic regression models using base splines [14].

- **Generalized Additive Models with Smoothing Splines and Generalized Additive Models with Natural Cubic Splines (`sgam` and `nsgam`)**: In addition to the basic spline structure, Generalized Additive Models are applied with natural cubic splines and smoothing splines using the `gam` library in R [15].

Possible interactions between the two diagnostic tests can also be considered within the non-linear approach. This may be advantageous, particularly if there is a correlation between these two markers. The `include.interact` option in the `nonlinComb` function can be set to `TRUE` to include interactions when building the model.

## 3.3 Mathematical operators:

- **Arithmetic Operators** : Arithmetic operators such as addition (`add`), subtraction (`subtract`), multiplication (`multiply`), and division (`divide`) can be used as mathematical operators within the `dtComb` package.

- **Distance Measures**: The combination of markers using these mathematical operators is evaluated based on distance measures, which assess the relationships between marker values [16, 17, 18]. The included distance measures with their respective formulas within the package are outlined as follows:

  - **Euclidean**(`euclidean`): $c_i = \sqrt{(x_{i1}-0)^2 + (x_{i2}-0)^2}$
  - **Manhattan**(`manhattan`): $c_i = |x_{i1}-0| + |x_{i2}-0|$
  - **Chebyshev**(`chebyshev`): $c_i = max|x_{i1}-0|, |x_{i2}-0|$
  - **Kulczynski d**(`kulczynski-d`): $c_i = \frac{|x_{i1}-0|+|x_{i2}-0|}{min(x_{i1},x_{i2})}$
  - **Lorentzian**(`lorentzian`): $c_i = (ln(1+|x_{i1}-0|)) + (ln(1+|x_{i2}-0|))$
  - **Taneja**(`taneja`): $c_i = z_1 \times \left( log\frac{z_1}{\sqrt{(x_{i1}\times\epsilon)}} \right) + z_2 \times \left( log\frac{z_2}{\sqrt{(x_{i2}\times\epsilon)}} \right)$
    where $z_1 = \frac{(x_{i1}-0)}{2}, z_2 = \frac{(x_{i2}-0)}{2}$
  - **Kumar-Johnson**(`kumar-johnson`): $c_i = \frac{(x_{i1}-0)^2}{2(x_{i1}\times\epsilon)} + \frac{(x_{i2}-0)^2}{2(x_{i2}\times\epsilon)}$, $\epsilon = 0.00001$
  - **Avg**(`avg`): $c_i = \frac{|x_{i1}-0|+|x_{i2}-0|+max(x_{i1}-0),(x_{i2}-0)}{2}$

- **Exponential approach**: This method combines diagnostic tests to examine relationships between diagnostic measurements (i.e., markers). In this approach, one of the two diagnostic tests is considered the base, and the other is an exponent. This relationship is denoted by the terms `baseinexp` ($x_{i1}^{x_{i2}}$) and `expinbase` ($x_{i2}^{x_{i1}}$), respectively.

To increase the performance of the diagnostic test results, one can transform the values of markers before applying mathematical operators. It is possible to apply transformations like *cosine* (`cos`), *sine* (`sin`), *exponential* (`exp`), and *logarithmic* (`log`). Similarly, when using add and subtract operators, the exponents of markers are iteratively adjusted by 0.1 within the range [-3, 3]. This adjustment aims to optimize the **AUC**, and the model with the highest **AUC** is chosen as the final model.

### 3.4 Machine-learning algorithms:

Given that the diagnostic test data consists of numerical inputs and aims to predict binary outcomes, we selected 113 models from the `caret` package that meet these criteria. We benefit from these 113 models to create the `mlComb` function, which combines diagnostic tests using machine-learning algorithms. For a list of machine learning algorithms included in the `dtComb` package, users can run the `availableMethods` function [19].

## 4 Standardization

Standardization is critical in data analysis, especially when dealing with variables with different units or scales. Standardization plays a vital role in ensuring fair comparisons and accurate modeling in the context of diagnostic tests containing multiple variables measured in different units. In *dtComb*, while standardization is optional, certain combination methods within the *dtComb* package such as `minmax`, `PCL`, `PT` enforce standardization by default. For linear and non-linear combination methods and mathematical operators, five different standardization methods are available, listed as follows:

- **Z-score**: This method scales the data to have a mean of 0 and a standard deviation of 1. It subtracts the mean and divides by the standard deviation for each feature. Mathematically,

$$Z - score = \frac{x - (\overline{x})}{sd(x)}$$

  where $x$ is the value of a marker, $\overline{x}$ is the mean of the marker, and $sd(x)$ is the standard deviation of the marker.

- **T-score**: T-score is commonly used in data analysis to transform raw scores into a standardized form. The standard formula for converting a raw score $x$ into a T-score is:

$$T - score = \left(\frac{x - (\overline{x})}{sd(x)} \times 10\right) + 50$$

  where $x$ is the value of a marker, $\overline{x}$ is the mean of the marker, and $sd(x)$ is the standard deviation of the marker.

- **Range (a.k.a. min-max scaling)**: This method transforms data to a specific range between 0 and 1. The formula for this method is:

$$Range = \frac{x - min(x)}{max(x) - min(x)}$$

- **Mean**: This method, which helps to understand the relative size of a single observation concerning the mean of the dataset, calculates the ratio of each data point to the mean value of the dataset.

$$Mean = \frac{x}{\overline{x}}$$

  where $x$ is the value of a marker and $\overline{x}$ is the mean of the marker.

- **Deviance**: This method, which allows for the comparison of individual data points about the overall spread of the data, calculates the ratio of each data point to the standard deviation of the dataset.

$$Deviance = \frac{x}{sd(x)}$$

  where $x$ is the value of a marker and $sd(x)$ is the standard deviation of the marker.

The `mlComb` function, designed for combining two diagnostic tests using machine-learning approaches, leverages the diverse set of standardization methods provided by the `caret` package. This empowers users to choose the optimal method tailored to their data and model needs. For guidance on default standardization methods or specifying particular standardization techniques for different models, users can refer to the `caret` documentation.

# 5 Model building

The dtComb has four different functions (`linComb`, `nonlinComb`, `mlComb`, `mathComb`) for the model building and evaluation process. These functions can be used to evaluate selected model providing a set of values for the model parameters, return the optimal model as well as the overall performance of the model for the training set.

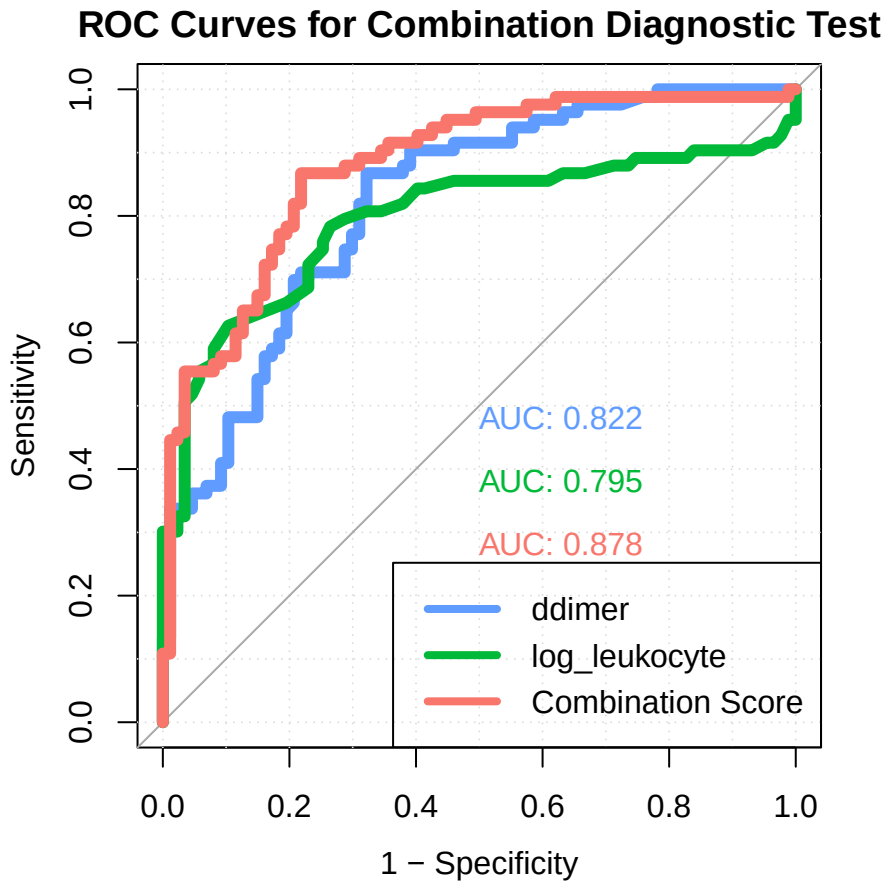
## 5.1 Resampling methods to optimize the model parameters

The dtComb package optimizes model parameters for linear and non-linear approaches by employing various validation techniques: (i) n-fold cross-validation, which involves splitting the training data into `nfolds` groups for the model assessment, (ii) 10-fold repeated cross-validation where 10-fold division is repeated `nrepeat` times to ensure robust model evaluation and (iii) bootstrapping which makes use of `niters` subgroups from the training dataset to enhance parameter optimization and model validation. The resampling function embedded within the `caret` package is used by the mlComb function to perform resampling and hyper-parameter optimization. The relevant section of the `caret` package documentation contains detailed information about this process [19].

## 5.2 Model evaluation in the training phase

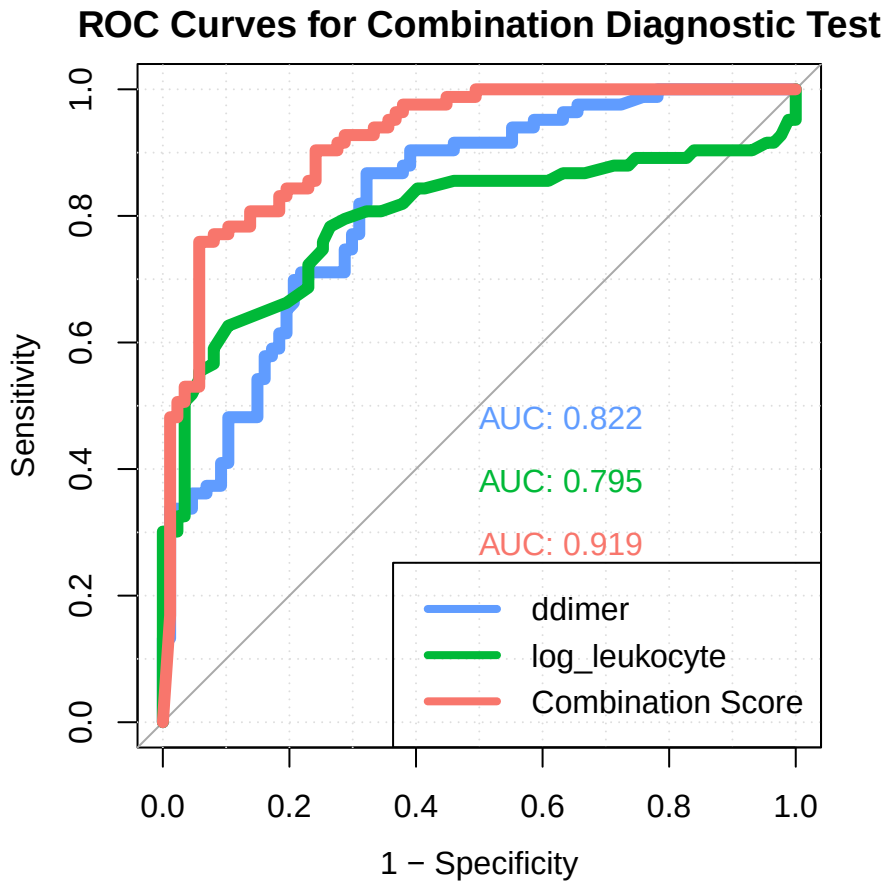
Metrics such as Receiver Operating Characteristic (ROC) curves and Area Under the Curve (AUC) values evaluate the model's performance during training. These metrics provide insights into the model's ability to distinguish between different classes or categories, offering valuable information regarding its performance characteristics. While measuring the ROC curves, an argument called `direction` argument is given as input to the relevant function, and the default value of the `direction` is set to `auto`. Moreover, the cut-off point which determines AUC is controlled by the `cutoff.method` argument within the package. 34 methods are available to determine the cut-off point, accessible through the `OptimalCutpoints` package in R [20]. Now, we will provide examples of how to use each approach's specific functions separately. We selected the cutoff.method for each scenario as the `Youden Index` and specified the ROC curve's direction as `<`.
Using the training data provided earlier, for a linear combination approach, the `linComb` function is employed with the `range`. Standardization method and 5-fold cross-validation in the following manner:

```
set.seed(2128)

# linComb Function
fit.lin <- linComb(
  markers = markers,
  status = status,
  event = "needed",
  method = "scoring",
  resample = "cv",
  standardize = "range",
  ndigits = 2, direction = "auto",
  cutoff.method = "Youden"
)
```



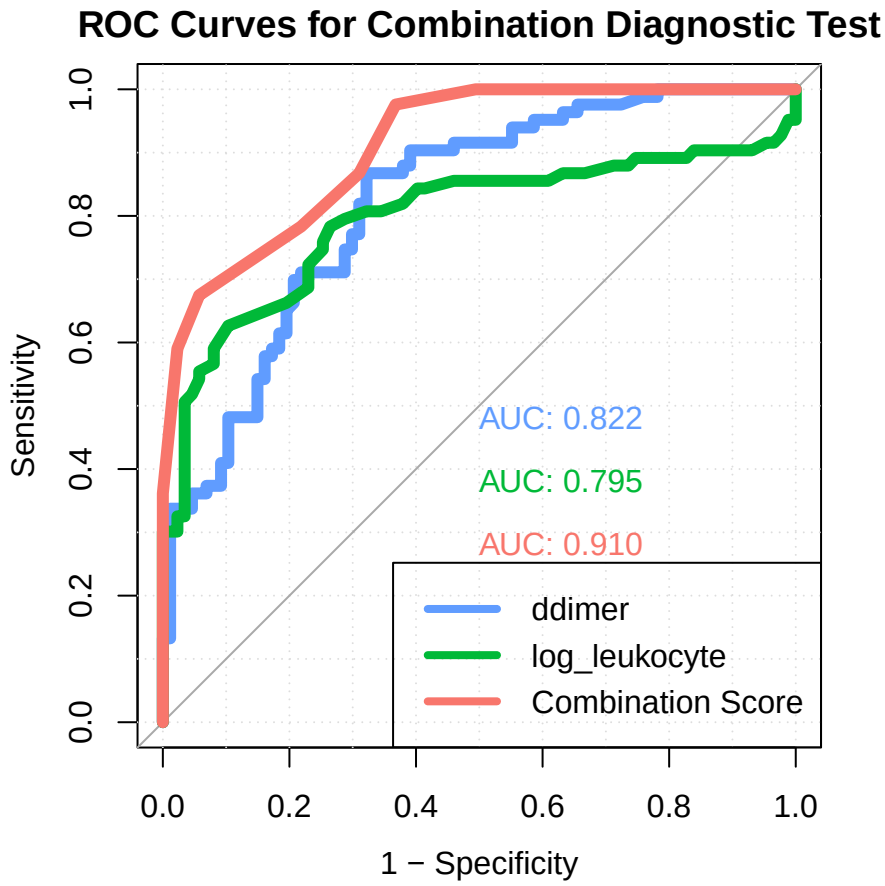

Let us now assume that we aim to fit the same training data using the Lasso regression method, which falls under the category of non-linear approaches. We'll use the `nonlinComb` function for a non-linear combination method, incorporating the bootstrapping resampling method with `niter=10`, and specifying additional arguments as follows:

```
# nonlinComb Function
set.seed(2128)

fit.nonlin <- nonlinComb(
  markers = markers,
  status = status,
  event = "needed",
  method = "lassoreg",
  include.interact = "TRUE",
  resample = "boot",
  direction = "auto",
  cutoff.method = "Youden"
)
```



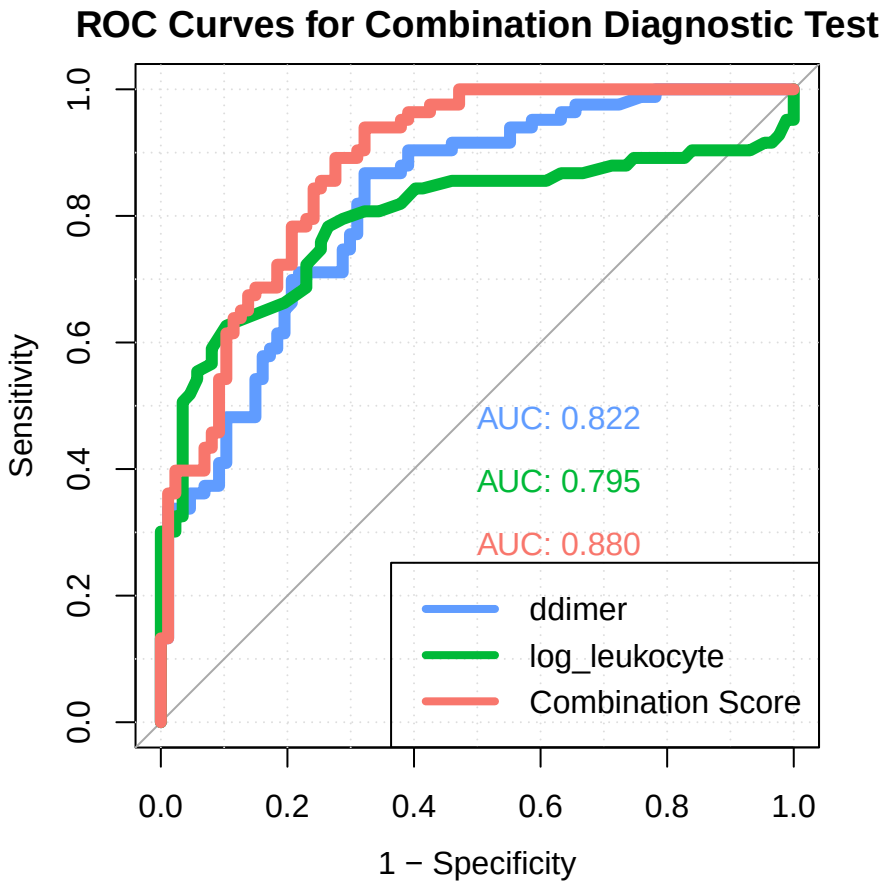

In the following example, we fit the training data using the `knn` (K-Nearest Neighbors) method, a machine-learning approach. We will use the `mlComb` function, incorporating the 10-folds repeated cross-validation technique (i.e.,`nfolds = 10`, `nrepeats = 5`) as follows:

```
# mlComb Function
set.seed(2128)

fit.ml <- mlComb(
  markers = markers,
  status = status,
  event = "needed",
  method = "knn",
  resample = "repeatedcv", nfolds = 10, nrepeats = 5,
  preProcess = c("center", "scale"),
  direction = "<", cutoff.method = "Youden"
)
```




In the final example, we'll implement the `mathComb` function, specifically designed for mathematical operators. Using the same training dataset as in the previous examples, the chosen method involves utilizing the `Euclidean` distance metric to train the model as follows:

```
# mathComb Function
fit.math <- mathComb(
  markers = markers,
  status = status,
  event = "needed",
  method = "distance",
  distance = "euclidean",
  direction = "<",
  cutoff.method = "Youden"
)
```




The results of the four described approaches and single diagnostic tests are summarized in Table 1. The findings indicate that the combined diagnostic tests outperformed the individual ones. Notably, the `Lasso regression` method had the highest AUC value among the combined approaches. In this vignette, we compared only a few models and demonstrated how to train models. Acknowledging that different data and models might yield different results is essential. We will use the model trained by the `Lasso regression` method to make predictions on the test set since it exhibited superior performance on the training set.

Table 1: Combination results for train data

| Metot | AUC | Accuracy |
|---|---|---|
| D-dimer | 0.822 | 0.77 |
| log(leukocyte) | 0.795 | 0.77 |
| scoring | 0.878 | 0.82 |
| lassoreg | 0.919 | 0.85 |
| knn | 0.910 | 0.81 |
| distance(euclidean) | 0.880 | 0.81 |

## 6 Predicting the class labels of test samples

We use the model parameters obtained during the training phase to predict the class labels of test samples. For instance, when training a model using the `Lasso regression` method, the labels of the test set are predicted based on the parameters optimized during training. However, the test set must undergo the same standardization or preprocessing steps as the training set to ensure both sets are on the same scale

before making predictions. The `predict` function is then applied to the standardized test samples to estimate the class label (status) of new samples, as shown below:

```
predict(fit.nonlin, testData)
```

```
##     comb.score     labels
## 1  1.000000000     needed
## 2  0.996516222     needed
## 3  0.999999897     needed
## 4  0.623530874     needed
## 5  1.000000000     needed
## 6  0.986539360     needed
## 7  0.999999625     needed
## 8  1.000000000     needed
## 9  0.999945221     needed
## 10 0.683216423     needed
## 11 0.999822666     needed
## 12 0.079077514 not_needed
## 13 0.599066868     needed
## 14 0.460615670 not_needed
## 15 0.525039302 not_needed
## 16 1.000000000     needed
## 17 0.983148471     needed
## 18 1.000000000     needed
## 19 0.874697566     needed
## 20 1.000000000     needed
## 21 0.332037411 not_needed
## 22 0.073757039 not_needed
## 23 0.378202968 not_needed
## 24 1.000000000     needed
## 25 0.999836211     needed
## 26 0.324951786 not_needed
## 27 0.734074461     needed
## 28 0.016074836 not_needed
## 29 0.724199018     needed
## 30 0.628914315     needed
## 31 0.274151673 not_needed
## 32 0.329407757 not_needed
## 33 0.127206060 not_needed
## 34 0.028066565 not_needed
## 35 0.123377049 not_needed
## 36 0.093700644 not_needed
## 37 0.050466767 not_needed
## 38 0.074684122 not_needed
## 39 0.399087783 not_needed
## 40 0.499194405 not_needed
## 41 0.006978096 not_needed
## 42 0.017378012 not_needed
## 43 0.386983215 not_needed
## 44 0.339655425 not_needed
## 45 0.331853431 not_needed
## 46 1.000000000     needed
## 47 0.349232483 not_needed
## 48 0.430754820 not_needed
## 49 0.409973137 not_needed
## 50 0.369327194 not_needed
```

```
## 51 0.005090672 not_needed
## 52 0.592396072     needed
## 53 0.533448209 not_needed
## 54 0.025102711 not_needed
## 55 0.002066089 not_needed
```

When employed on models trained with the `linComb` or `mathComb` functions, the `predict` function returns the combination score of the applied method and the estimated label. The `predict` function, on the other hand, returns the probability of positive and negative cases for each test observation for models trained with the `nonlinComb` or `mlComb` function.

# 7 Session info

```
sessionInfo()

## R version 4.3.1 (2023-06-16)
## Platform: aarch64-apple-darwin20 (64-bit)
## Running under: macOS Sonoma 14.6.1
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRlapack.dylib;  LAPACK
##
## locale:
## [1] C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: Europe/Istanbul
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] caret_6.0-94   lattice_0.22-5 ggplot2_3.5.1  dtComb_1.0.4   knitr_1.46.3
##
## loaded via a namespace (and not attached):
##   [1] DBI_1.2.1               pROC_1.18.5             BiasedUrn_2.0.11
##   [4] rlang_1.1.4             magrittr_2.0.3          e1071_1.7-14
##   [7] compiler_4.3.1          systemfonts_1.0.5       vctrs_0.6.5
##  [10] reshape2_1.4.4          stringr_1.5.1           httpcode_0.3.0
##  [13] shape_1.4.6             pkgconfig_2.0.3         crayon_1.5.3
##  [16] fastmap_1.1.1           ellipsis_0.3.2          pander_0.6.5
##  [19] utf8_1.2.4              promises_1.2.1          rmarkdown_2.25
##  [22] prodlim_2023.08.28      epiR_2.0.67             ragg_1.2.7
##  [25] purrr_1.0.2             glmnet_4.1-8            xfun_0.43
##  [28] jsonlite_1.8.8          recipes_1.0.9           highr_0.10
##  [31] later_1.3.2             uuid_1.2-0              parallel_4.3.1
##  [34] R6_2.5.1                stringi_1.8.4           parallelly_1.36.0
##  [37] rpart_4.1.23            lubridate_1.9.3         Rcpp_1.0.13
##  [40] iterators_1.0.14        future.apply_1.11.1     zoo_1.8-12
##  [43] httpuv_1.6.14           Matrix_1.6-5            splines_4.3.1
##  [46] nnet_7.3-19             timechange_0.2.0        tidyselect_1.2.1
##  [49] rstudioapi_0.15.0       timeDate_4032.109       codetools_0.2-19
##  [52] curl_5.2.0              listenv_0.9.0           tibble_3.2.1
```

```
##  [55] plyr_1.8.9              shiny_1.8.0             withr_3.0.1
##  [58] askpass_1.2.0           flextable_0.9.4         evaluate_0.23
##  [61] OptimalCutpoints_1.1-5  future_1.33.1           survival_3.5-7
##  [64] sf_1.0-15               units_0.8-5             proxy_0.4-27
##  [67] zip_2.3.0               xml2_1.3.6              pillar_1.9.0
##  [70] KernSmooth_2.23-22      foreach_1.5.2           stats4_4.3.1
##  [73] generics_0.1.3          munsell_0.5.1           scales_1.3.0
##  [76] globals_0.16.2          xtable_1.8-4            class_7.3-22
##  [79] glue_1.7.0              gdtools_0.3.5           tools_4.3.1
##  [82] gfonts_0.2.0            data.table_1.16.0       ModelMetrics_1.2.2.2
##  [85] gower_1.0.1             grid_4.3.1              ipred_0.9-14
##  [88] colorspace_2.1-1        nlme_3.1-164            cli_3.6.3
##  [91] textshaping_0.3.7       officer_0.6.3           fontBitstreamVera_0.1.1
##  [94] fansi_1.0.6             lava_1.7.3              dplyr_1.1.4
##  [97] gtable_0.3.5            digest_0.6.34           fontquiver_0.2.1
## [100] classInt_0.4-10         crul_1.4.0              htmltools_0.5.7
## [103] lifecycle_1.0.4         hardhat_1.3.0           mime_0.12
## [106] fontLiberation_0.1.0    openssl_2.1.1           MASS_7.3-60.0.1
```

# References


[1] Maedeh Amini, Anoshirvan Kazemnejad, Farid Zayeri, Azam Amirian, and Nourossadat Kariman. Application of adjusted-receiver operating characteristic curve analysis in combination of biomarkers for early detection of gestational diabetes mellitus. *Koomesh*, 2019.

[2] Suizhi Yu. A covariate-adjusted classi cation model for multiple biomarkers in disease screening and diagnosis. *Kansas State University*, 2019.

[3] Rocío Aznar-Gimeno, Luis M Esteban, Gerardo Sanz, Rafael del Hoyo-Alonso, and Ricardo Savirón-Cornudella. Incorporating a new summary statistic into the min–max approach: a min–max–median, min–max–iqr combination of biomarkers for maximising the youden index. *Mathematics*, 9(19):2497, 2021.

[4] Hizir Yakup Akyildiz, Erdogan Sozuer, Alper Akcan, Can Kuçuk, Tarik Artis, İsmail Biri, Namık Yılmaz, et al. The value of d-dimer test in the diagnosis of patients with nontraumatic acute abdomen. *Turkish Journal of Trauma and Emergency Surgery*, 16(1):22–26, 2010.

[5] Strother H Walker and David B Duncan. Estimation of the probability of an event as a function of several independent variables. *Biometrika*, 54(1-2):167–179, 1967.

[6] Cristóbal León, Sergio Ruiz-Santana, Pedro Saavedra, Benito Almirante, Juan Nolla-Salas, Francisco Álvarez-Lerma, José Garnacho-Montero, María Ángeles León, EPCAN Study Group, et al. A bedside scoring system ("candida score") for early antifungal treatment in nonneutropenic critically ill patients with candida colonization. *Critical care medicine*, 34(3):730–737, 2006.

[7] Margaret Sullivan Pepe and Mary Lou Thompson. Combining diagnostic test results to increase accuracy. *Biostatistics*, 1(2):123–140, 2000.

[8] Margaret Sullivan Pepe, Tianxi Cai, and Gary Longton. Combining predictors for classification using the area under the receiver operating characteristic curve. *Biometrics*, 62(1):221–229, 2006.

[9] Chunling Liu, Aiyi Liu, and Susan Halabi. A min–max combination of biomarkers to improve diagnostic accuracy. *Statistics in medicine*, 30(16):2005–2014, 2011.

[10] John Q Su and Jun S Liu. Linear combinations of multiple diagnostic markers. *Journal of the American Statistical Association*, 88(424):1350–1355, 1993.

[11] G Sameera, R Vishnu Vardhan, and KVS Sarma. Binary classification using multivariate receiver operating characteristic curve for continuous data. *Journal of biopharmaceutical statistics*, 26(3): 421–431, 2016.

[12] Nicolae Todor, Irina Todor, and Gavril Săplăcan. Tools to identify linear combination of prognostic factors which maximizes area under receiver operator curve. *Journal of clinical bioinformatics*, 4(1): 1–7, 2014.

[13] Jerome Friedman, Trevor Hastie, and Rob Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1, 2010.

[14] R Core Team et al. R: A language and environment for statistical computing. 2013.

[15] Trevor Hastie. gam: Generalized additive models. r package version 1.16. 1. *Von https://CRAN. R-project. org/package= gam abgerufen*, 2019.

[16] Sung-Hyuk Cha. Comprehensive survey on distance/similarity measures between probability density functions. *City*, 1(2):1, 2007.

[17] Shraddha Pandit, Suchita Gupta, et al. A comparative study on distance measuring approaches for clustering. *International journal of research in computer science*, 2(1):29–31, 2011.

[18] Georgy Minaev, Robert Piché, and Ari Visa. Distance measures for classification of numerical features. *Tampere University of Technology, Finland*, 2018.

[19] Max Kuhn. Building predictive models in r using the caret package. *Journal of statistical software*, 28:1–26, 2008.

[20] Mónica López-Ratón, María Xosé Rodríguez-Álvarez, Carmen Cadarso-Suárez, and Francisco Gude-Sampedro. Optimalcutpoints: an r package for selecting optimal cutpoints in diagnostic tests. *Journal of statistical software*, 61:1–36, 2014.