Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *doi: 10.18637/jss.v000.i00*

EPX: Ensemble of Classifiers Based on Phalanxes of Variables for Highly Unbalanced Binary Classification Problems

Grace G. Hsu University of British Columbia

Jabed H. Tomal Thompson Rivers University

William J. Welch University of British Columbia

Abstract

In binary classification problems with a rare class of interest, there is by definition relatively little data for the rare class with which to build models. On the other hand, the number of useful variables for classification could be high-dimensional. For example, in drug discovery, there are usually very few bioactive compounds in large chemical libraries, whereas thousands of potentially useful explanatory variables characterize a compound's chemical structure. The sparsity of data for the class of interest makes it difficult for standard classification methods to exploit the richness of the useful explanatory variables. In contrast, the algorithm proposed by Tomal *et al.* (2015, 2016, 2019) builds an ensemble of classifiers from several disjoint and diverse subsets of explanatory variables, called *phalanxes*, and hence outstrips the predictive performance of many competitive state-of-the-art methods. Here, we present the R package **EPX** which implements the algorithm to form the ensemble of phalanxes as well as associated functions.

Keywords: Classification, Ensemble learning, Machine learning, Phalanxes, R.

1. Introduction

The ensemble of phalanxes (EPX) classifier is the result of the algorithm proposed by Tomal *et al.* (2015, 2016, 2019) and motivated by applications where the goal is statistical detection of a rare class of objects in a two-class classification problem. The algorithm clusters the explanatory variables into disjoint subsets called *phalanxes* such that the phalanxes work well in the constituent models when aggregated together. By "working well," we mean by some measure of how highly the observations belonging to the desired rare class are ranked via their estimated probabilities. Variables in different phalanxes and hence models can contribute to the overall ensemble without working against each other by way of deselection. Altogether

with the relatively low-dimensionality of each phalanx, it follows that datasets with high-dimension can be exploited in the face of limited response information. For more details on existing methods for such binary classification problems please see Tomal *et al.* (2015, 2016, 2019).

The EPX classifier's better performance against the already competitive RF in the areas of QSAR studies (Tomal *et al.* 2015, 2016), and potential for application in other problems with little information in the response variable relative to the number of explanatory variables (Tomal *et al.* 2017, 2019) thus motivated the implementation of the phalanx-formation algorithm in R (R Core Team 2020) package **EPX**. The **EPX** package is available from the Comprehensive R Archive Network (CRAN) at: `https://CRAN.R-project.org/package=EPX`.

The rest of article is organised as follows. Section 2 briefly discusses the performance assessment metrics and summarises the phalanx formation algorithm detailed by Tomal *et al.* (2015). Section 3 introduces the **EPX** package starting with the `epx` function, which trains an EPX classifier. The remainder of the section describes the main aspects of the other functions in the package. Throughout, we follow an example of how the package may be used. Section 4 encloses some remarks on further capabilities of the package in terms of ways to customise aspects of the algorithm. Finally, section 5 concludes the article.

2. Building the EPX classifier

2.1. Performance measures

Consider a binary classification problem where the relevant class is extremely rare and $\hat{\pi}(\mathbf{x})$ estimates the true probability of relevance $\pi(\mathbf{x})$ given the vector of features $\mathbf{x}$. In these highly unbalanced classification problems, building $\hat{\pi}$ to minimise the misclassification rate is no longer appropriate because this often results in a trivial classifier that classifies all objects as irrelevant. Instead, the average hit rate is considered as a better criterion for evaluating the performance of classifiers for unbalanced classification problems (Wang 2005, Chapter 3).

The average hit rate (AHR) summarises a classifier's hit curve, which plots the number of relevant objects against the number of total objects selected, where object selection is based on their ranking according to $\hat{\pi}$. Given n objects ranked in descending order according to their estimated probabilities of relevance $\hat{\pi}$, let this ordered list of objects have the true response values $y_{(1)}, y_{(2)}, ..., y_{(n)}$, where

$$y_{(i)} = \begin{cases} 1, & \text{if the } i\text{th object is relevant} \\ 0, & \text{if the } i\text{th object is irrelevant,} \end{cases}$$

and $M = \sum_{i=1}^{n} y_{(i)}$ is the total number of relevant objects. The AHR is hence defined as

$$\text{AHR} = \frac{1}{M} \sum_{i=1}^{n} \left[y_{(i)} \sum_{j=1}^{i} \frac{y_{(j)}}{i} \right]. \tag{1}$$

Note that AHR has a definition equivalent to that of *average precision* and a classifier is judged as performing well when it ranks the truly relevant objects more highly. Furthermore, when the $\hat{\pi}$ values for all n objects are not unique, we have objects with the same ranking.

In such cases, we assume that the objects with tied ranks are in random order, thus allowing the calculation of the expected AHR in closed form (Wang 2005, Chapter 3).

While there are other performance measures to summarize a hit curve (further details in section 4), there are several advantageous properties of AHR detailed in Wang (2005, Chapter 3.4). Thus, AHR is the default performance measure in the **EPX** package.

2.2. Phalanx-formation algorithm

The main idea of the algorithm proposed by Tomal *et al.* (2015) is that it groups the explanatory variables into disjoint subsets called *phalanxes* such that the variables in each phalanx work better in terms of some performance measure together in a single model than in separate models, while also accounting for the performance of the ensemble of the models. The steps of phalanx-formation above and subsequent construction of the ensemble of phalanxes is summarised in Figure 1.

Suppose we have a training data set with n observations and D explanatory variables. The four steps of the phalanx-formation algorithm are as follows:

1. **Initial phalanxes**. The D explanatory variables are partitioned into d initial phalanxes where $1 < d \leq D$.

2. **Filtered phalanxes**. The d initial phalanxes are filtered to $s \leq d$ filtered phalanxes. Various comparisons are done between each phalanx and the reference distribution of some assessment criterion a, such as AHR, under random ranking. For the computation of the reference distribution see Tomal *et al.* (2015, Section 4). In summary,

   - a_α: The α quantile of the reference distribution of the assessment criterion a. Default is $\alpha = 0.95$.
   - $\hat{\pi}_i$: The estimated probabilities of relevance from the classifier built only with the variables in phalanx i.
   - $a_i = a\left(\hat{\pi}_i\right)$: The performance measure of phalanx i found by passing $\hat{\pi}_i$ through the assessment criterion a.
   - $\hat{\pi}_{ij}$: The estimated probabilities of relevance from the classifier built with all the variables in phalanx i as well as those in phalanx j $(i \neq j)$.
   - $a_{ij} = a\left(\hat{\pi}_{ij}\right)$: The performance measure of a phalanx that includes all the variables in phalanxes i and j $(i \neq j)$.
   - $a_{\overline{ij}} = a\left(\frac{\hat{\pi}_i + \hat{\pi}_j}{2}\right)$: The performance measure of an ensemble of two models built with phalanxes i and j, respectively.

   That is, phalanx i survives if it passes at least one of the following three tests.

   - Phalanx i performs well alone:

   $$a_i \geq a_\alpha. \tag{2}$$

   - Phalanx i improves the performance of another phalanx j when the two are used to build a single model:

   $$a_{0.5} + (a_{ij} - a_j) \geq a_\alpha. \tag{3}$$

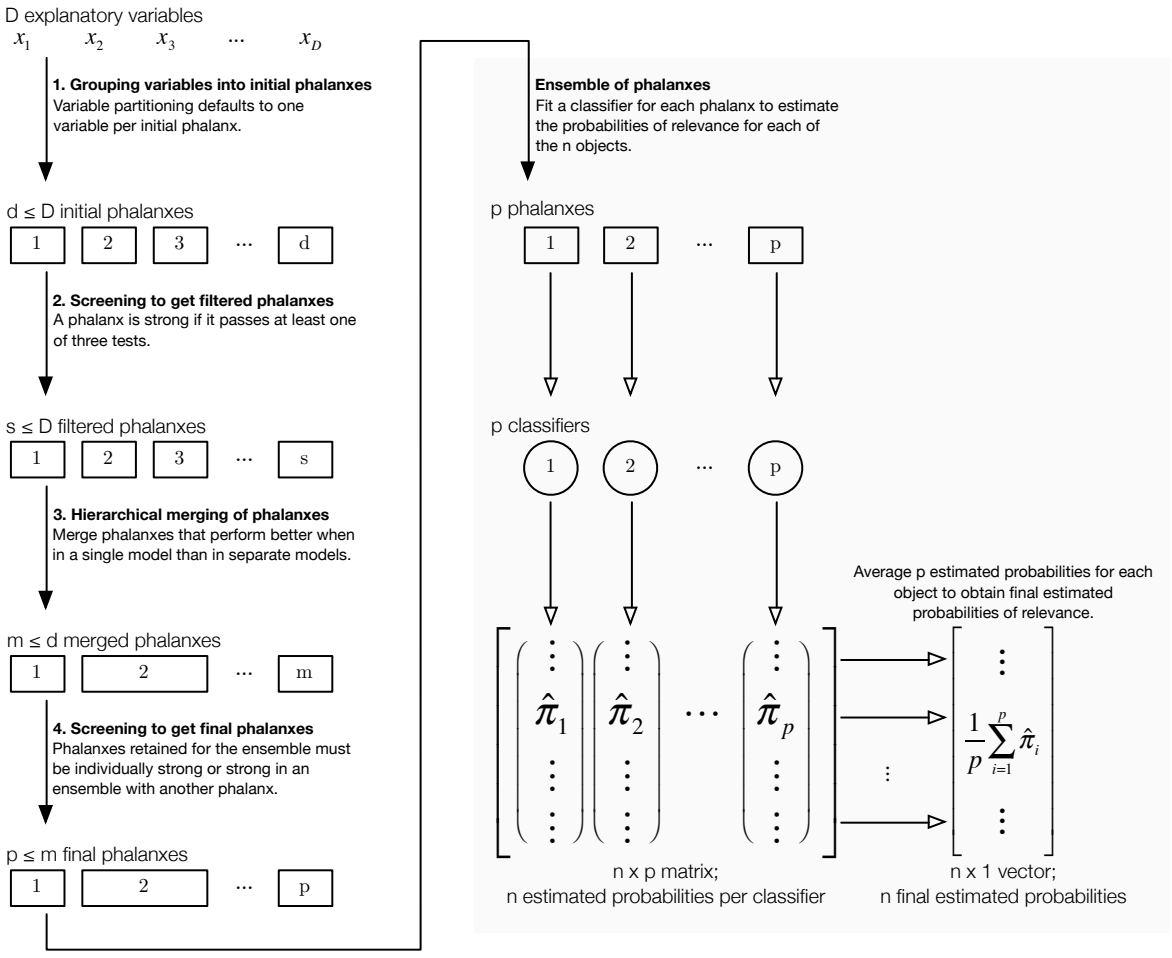

Figure 1: Algorithm for phalanx-formation and subsequent construction of the ensemble of the phalanxes. D variables are partitioned into d initial phalanxes, filtered to s phalanxes, merged into m phalanxes, and then filtered again to p final phalanxes used for the final ensemble ($D \geq d \geq s \geq m \geq p$).

- Phalanx i improves the performance of another phalanx j when the two are in an ensemble of two models:

$$a_{0.5} + \left(a_{\overline{ij}} - a_j\right) \geq a_\alpha. \tag{4}$$

The surviving s phalanxes are renamed accordingly from 1 to s.

3. **Merged phalanxes.** The s filtered phalanxes are merged hierarchically as follows: each iteration merges the pair of phalanxes i and j that minimises $m_{ij} = a_{\overline{ij}}/a_{ij}$. When $m_{ij} < 1$, this means that phalanxes i and j perform better in a single model than as an ensemble. Thus, each merge reduces the number of phalanxes by one until $m_{ij} \geq 1$ for all phalanxes i, j. We are left with $m \leq s$ phalanxes.

4. **Final phalanxes.** The m phalanxes are filtered in a way that a phalanx survives if

it performs well alone as in condition (2), or if it performs well in an ensemble with at least one other phalanx as in condition (4). Note that condition (3) is automatically satisfied due to the merging done in the previous step.

After obtaining the final p phalanxes, we construct the ensemble of phalanxes by building a classifier for each of the phalanxes. Each classifier then produces an n-length vector of estimated probabilities of relevance, resulting in p such vectors: $\hat{\pi}_1, \hat{\pi}_2, ..., \hat{\pi}_p$. The final vector of probabilities for the n objects are the result of averaging the p vectors of probabilities.

3. The EPX package

Training an EPX model is done by the `epx` function, where `x` and `y` are the explanatory variables contained in a data frame and the binary response variable vector (1 is the rare or relevant class) respectively. The main parameters of the `epx` are as follows:

- `phalanxes.initial`: designates the phalanx membership of each explanatory variable as a numeric vector. Defaults to one variable per phalanx.
- `classifier`: indicates what models we fit for each phalanx for the model as a string. Defaults to *random forest*.
- `classifier.args`: allows for some modification of the arguments for the choice of classifier, such as the number of trees allowed per random forest. Leaving an empty list means that the classifier will use its default settings.
- `performance`: indicates the choice of performance measure. Defaults to AHR.
- `computing`: indicates whether to compute in *parallel* and requires the user to register a parallel backend. Defaults to sequential computing, which registers a sequential backend.

Other parameters in `epx` are detailed in the documentation and allow for the tuning of the performance measure if appropriate, as well as some options regarding the filtering steps. These additional parameters all have defaults so we focus on the main arguments for the full example in this section.

We demonstrate the **EPX** package via an example of QSAR modelling from drug discovery, where the activity of a chemical compound is related to the compound's molecular structure. In this case an active compound is the desired relevant, the rare class, and the compound's molecular structure are characterised by various descriptors. One such descriptor set of explanatory variables are Burden numbers (BN) (Burden 1989). The BN descriptor set is one of multiple descriptor sets used in Tomal *et al.* (2015, 2016) and all its explanatory variables are continuous. Typically the data used to train an EPX model are of a size where parallel computing should be done to complete the phalanx-formation algorithm in a timely manner. This is because the computational complexity of phalanx formation is $O(d^2)$, where d is the number of initial phalanxes. However, for the sake of demonstration we work with an approximately 20% sample from the BN data, which has nearly 5000 observations, so that sequential computing is reasonably fast. In our balanced sample (`BNsample`) of size 1000, 990 observations are of inactive compounds ($y = 0$) and the rest 10 are active compounds ($y = 1$), thus preserving the roughly 1% prevalence of active compounds in the full BN descriptor set.

```
1 set.seed(761)
2 model <- epx(x = BNsample[,-25], y = BNsample[,25], classifier.args = list(ntree = 150))
```

Note that since we use the default random forest as our base classifier, we set a seed for reproducibility. To further cut down on runtime, we lower the number of trees from the default 500 to 150. While the algorithm is running we get some indication of progress in the console:

```
3 Performance measure: AHR
4 Performance measure additional arguments: none
5
6 Base classifier: random forest
7 Base classifier arguments specified in phalanx-formation:
8 $ntree
9 [1] 150
10
11 Phalanx formation is in progress, please wait
12
13 Reference distribution quantiles:
14 a0.95 = 0.0322050977060676
15 a0.50 = 0.0125942193317316
16
17
18 Number of deleted phalanxes after first filtering: 9 out of 24
19 Number of deleted phalanxes after final filtering: 0 out of 4
```

Lines $3-9$ summarise the crucial performance and classifier settings used in the algorithm. Line 11 tells the user that the phalanx formation is in progress. Lines $14-15$ display the values of a_α ($\alpha = 0.95$ by default) and $a_{0.5}$ respectively as described in section 2.2. Finally, lines $18-19$ display the results of the two filtering steps as each stage is completed (the second and fourth step in Figure 1). After about a minute, `epx` outputs an `S3` object of class "`epx`".

```
20 > str(model)
21 List of 8
22  $ PHALANXES                  :List of 4
23   ..$ phalanxes.initial : num [1:24] 1 2 3 4 5 6 7 8 9 10 ...
24   ..$ phalanxes.filtered: num [1:24] 1 2 3 0 4 5 6 7 8 0 ...
25   ..$ phalanxes.merged  : num [1:24] 1 1 1 0 1 1 2 1 1 0 ...
26   ..$ phalanxes.final   : num [1:24] 1 1 1 0 1 1 2 1 1 0 ...
27  $ PHALANXES.FINAL.PERFORMANCE: num [1:4] 0.0845 0.1022 0.0646 0.101
28  $ PHALANXES.FINAL.FITS       : num [1:1000, 1:4] 0 0 0 0 0 0 0 0 0 0 ...
29   ..- attr(*, "dimnames")=List of 2
30   .. ..$ : NULL
31   .. ..$ : NULL
32  $ ENSEMBLED.FITS             : num [1:1000] 0 0 0 0 0 ...
33 > # 4 more items omitted
```

An "epx" object is a list starting with information regarding the phalanx membership of all the explanatory variables in all four steps of the phalanx-formation algorithm (lines $23-26$). Every distinct positive integer is the "name" of a phalanx and so being assigned to 0 indicates that the variable does not belong to any phalanx at all.

```
34 > model$PHALANXES
35 $phalanxes.initial
36  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
37
38 $phalanxes.filtered
39  [1]  1  2  3  0  4  5  6  7  8  0  9  0  10  0  11  0 12  0 13  0 14 0 15 0
40
41 $phalanxes.merged
42  [1] 1 1 1 0 1 1 2 1 1 0 1 0 1 0 3 0 2 0 1 0 2 0 4 0
43
44 $phalanxes.final
45  [1] 1 1 1 0 1 1 2 1 1 0 1 0 1 0 3 0 2 0 1 0 2 0 4 0
```

In our example, we begin with the default initial phalanxes where each variable is in its own phalanx (line 36). After the filtering, explanatory variables 4, 10, 12, 14, 16, 18, 20, 22, 24 did not survive and are hence assigned 0 values (line 39). Note that those variables that did survive filtering are renamed/renumbered such that the maximum value in each vector is equal to the total number of phalanxes. After merging (line 42), the 15 phalanxes have been reduced to *four* phalanxes, and finally in the last step of the algorithm (line 45), phalanxes $1, 2, 3, 4$ from line 49 survive final filtering, leaving a final count of *four* phalanxes.

Other items in an "epx" object include

- `PHALANXES.FINAL.PERFORMANCE` (line 27): the performance of each of the final phalanxes (indices of the vector match the phalanx) according to the chosen performance measure.
- `PHALANXES.FINAL.FITS` (lines $28-31$): the $n \times p$ (p is the number of final phalanxes) matrix of estimated probabilities of relevance where the ith column is the estimated probabilities from phalanx i, which matches the matrix shown in Figure 1. Here we have $n = 1000$, $p = 4$.
- `ENSEMBLED.FITS` (line 32): an n-length vector of final estimated probabilities, obtained by averaging across the columns of `PHALANXES.FINAL.FITS`. This is also represented in Figure 1 as a column vector of $\hat{p}_i$'s.

The four omitted items in an "epx" object consist of the data used to train the model, as well as information regarding base classifier and performance measure choices, which are required for other functions in the package described in the following sections.

3.1. Summary and plotting

Having fitted an EPX model, passing the resulting "epx" class object to the `summary` function provides an easily legible summary of the results of phalanx-formation along with the performance of each final phalanx (lines $46-66$).

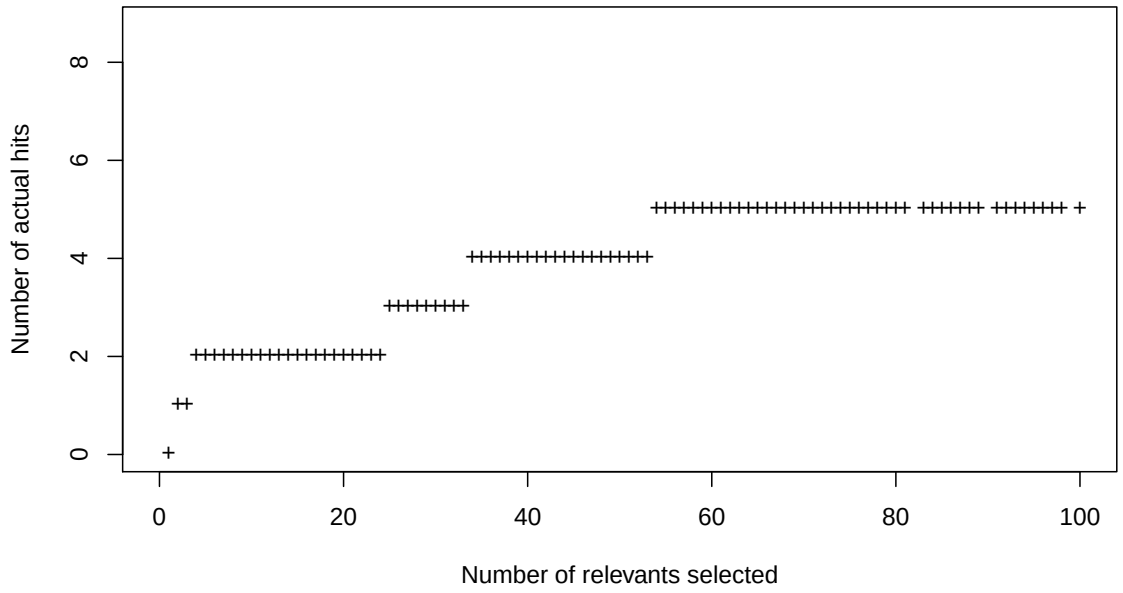

Figure 2: The hit curve generated by calling `plot(model)`.

```
> summary(model)
Phalanx-formation algorithm starts with 24 variable(s)
and ends with 15  variable(s) grouped into 4 phalanxes:
1 2 3 4
------
Phalanx 1 contains 10 variable(s):
WBN_GC_L_0.25, WBN_GC_H_0.25, WBN_GC_L_0.50, WBN_GC_L_0.75, WBN_GC_H_0.75, WBN_GC_H_1.00,
WBN_EN_L_0.25, WBN_EN_L_0.50, WBN_EN_L_0.75, WBN_LP_L_0.50
=> AHR is 0.08453896
------
Phalanx 2 contains 3 variable(s):
WBN_GC_L_1.00, WBN_LP_L_0.25, WBN_LP_L_0.75
=> AHR is 0.103273
------
Phalanx 3 contains 1 variable(s):
WBN_EN_L_1.00
=> AHR is 0.06456355
------
Phalanx 4 contains 1 variable(s):
WBN_LP_L_1.00
=> AHR is 0.1009757
```

Passing the "`epx`" object to the `plot` function generates a plot of the corresponding hit curve as seen in Figure 2. The hit curve is a step-function which increases by one step when one hit is found. On the other hand, the numbers of hits found may remain fixed for a while as

the number of shortlisted compounds grows.

The hit curve shows the ranking performance depending on when we cut-off the ranked list starting from the highest rank. A hit curve is superior to another hit curve if all its points lie above those of the other. Note that AHR is a numerical criterion that summarises a hit curve and in Figure 2, we see that the hit curve reflects the performance of the model fit by `epx`.

We also provide a `hit.curve` function that generates a hit curve in a given vector of predicted probabilities of relevance and its corresponding true response vector.

3.2. Prediction

Passing an "`epx`" object to the `predict` function without providing data for the `newdata` argument simply returns the `ENSEMBLED.FITS` element of the object as demonstrated in lines 67 – 76.

```
67 > preds0 <- predict(model)
68 Base classifier: random forest
69 Base classifier arguments specified in phalanx-formation:
70 $ntree
71 [1] 150
72
73 Base classifier arguments specified in prediction: none
74
75 > all.equal(preds0, model$ENSEMBLED.FITS)
76 [1] TRUE
```

However, if we provide some additional arguments for the classifier, `predict` will refit the classifiers for each phalanx as desired using the training data, thus resulting in different predictions than the "`epx`" object's `ENSEMBLED.FITS` (lines 77 – 89).

```
77 > set.seed(761)
78 > preds <- predict(model, classifier.args = list(ntree = 500))
79 Base classifier: random forest
80 Base classifier arguments specified in phalanx-formation:
81 $ntree
82 [1] 150
83
84 Base classifier arguments specified in prediction:
85 $ntree
86 [1] 500
87
88 > all.equal(preds, model$ENSEMBLED.FITS)
89 [1] "Mean relative difference: 1.121363"
```

Of course, we may also provide new data using the usual `predict` function framework and `predict` will output a vector of estimated probabilities using the trained EPX model. Here we use the remaining observations in the full BN descriptor set that were not included in `BNsample` to serve as the test data `BNhold`. Note that as usual we may use the same base

classifier arguments as those used in phalanx formation (line 91) or we may make some changes (lines $94 - 95$).

```
90 set.seed(761)
91 predshold0 <- predict(model, newdata = BNhold[,-25])
92
93 set.seed(761)
94 predshold <- predict(model, newdata = BNhold[,-25],
95                       classifier.args = list(ntree = 500))
```

3.3. Cross-validation

The `cv.epx` function performs balanced $k-$fold cross-validation given an "`epx`" class object. Cross-validation is a way of assessing the performance of the ensemble of phalanxes, and we use balanced cross-validation because we expect the training data to have a rare class. If we use random forest as the base classifier, for each phalanx i we have out-of-bag error with which we can estimate the performance of the ith phalanx. However, there is no clear way to combine the out-of-bag error values for all phalanxes and hence the need for a way to assess the performance of the phalanxes together as an ensemble.

The setup is as follows: the training data with n observations is randomly divided into k approximately equal-sized groups (folds), where each of the k folds will have approximately $1/k$ of the relevant observations. We train an EPX model based on the given "`epx`" class object using $k - 1$ of the folds, leaving one fold out to serve as the test set. This is repeated until all folds have served as a test set, at which point we will have final estimated probabilities of relevance for all n observations.

Note that we do not use the $k - 1$ folds to rerun the phalanx-formation algorithm, instead always using the (final) phalanxes specified in the "`epx`" object. Hence, this is biased cross-validation and motivated by the generally exorbitant amount of time required for phalanx-formation to be done k times.

The `cv.epx` function by default performs $10-$fold balanced cross-validation ($k = 10$) with observations randomly divided into folds. As with the `predict` function, `cv.epx` must take an "`epx`" object.

```
96 > set.seed(761)
97 > cv150 <- cv.epx(model)
98 Base classifier: random forest
99 Base classifier arguments specified in phalanx-formation:
100 $ntree
101 [1] 150
102
103 Base classifier arguments specified in balanced 10-fold cross-validation: none
```

The text in the console will indicate both the number of folds and any arguments made regarding the base classifier for cross-validation (line 101). This is similar to the behaviour of `predict` in that we still fit the same base classifier as that used during phalanx-formation, but we may change its arguments via the `classifier.args` argument.

```
> set.seed(761)
> cv500 <- cv.epx(model, classifier.args = list(ntree = 500))
Base classifier: random forest
Base classifier arguments specified in phalanx-formation:
$ntree
[1] 150

Base classifier arguments specified in balanced 10-fold cross-validation:
$ntree
[1] 500
```

The output of `cv.epx` by default is an $(n+1) \times (p+1)$ matrix, where n is the number of observations and p is the number of phalanxes. The ith column of the matrix is the predicted probabilities of relevance from the ith phalanx except the last $(p+1)$th column, which has the predicted probabilities of relevance from the ensemble of phalanxes. The $(n+1)$th row of the matrix is the performance of the corresponding column, determined by the performance measure specified by the "`epx`" object (line 121).

```
> tail(cv500)
                      1         2          3          4  ensemble
            0.45000000 0.0760000 0.66800000 0.00400000 0.2995000
            0.00000000 0.0000000 0.00000000 0.00000000 0.0000000
            0.00000000 0.0000000 0.00000000 0.00000000 0.0000000
            0.07200000 0.0000000 0.00000000 0.00000000 0.0180000
            0.00000000 0.0000000 0.00000000 0.00000000 0.0000000
performance 0.07805713 0.1039471 0.05997207 0.09088224 0.1457195
```

Some additional arguments of `cv.epx` are briefly summarised follows:

- `folds`: allows the use of custom folds by taking an n-length vector of positive integer values from 1 to k, such that the ith value in the vector indicates to which fold the ith observation should be assigned.
- `folds.out`: provides an n-length vector recording the fold membership for each observation; default is `FALSE`. Setting as `TRUE` changes the output of `cv.epx` into a list of two elements. The first being the $(n+1) \times (p+1)$ matrix attained from `cv.epx` by default, and the second is the $n-$length vector with integer values from 1 to k indicating to which fold each observation was shuffled for cross-validation.

3.4. Example with parallel computing

Since the computational complexity of phalanx formation is $O(d^2)$ where d is the number of initial phalanxes, often we need to run the `epx` function in parallel. The **EPX** package uses the **foreach** package to parallelise the loops where a_i, a_{ij}, and $a_{\overline{ij}}$ are calculated (i, j are phalanxes) (Analytics and Weston 2020). Here we use **doParallel** (Revolution Analytics and Weston 2019) to demonstrate a way to register a parallel backend for the `%dopar%` function from **foreach**.

Generally the setup for parallel computing is as follows:

```
clusters <- parallel::detectCores()      # number of jobs simultaneously
cl <- parallel::makeCluster(clusters)    # make clusters
doParallel::registerDoParallel(cl)               # use the above clusters

########################################################################
############ Functions that use parallel computing run here ############
########################################################################

parallel::stopCluster(cl)                # close clusters
```

In the following example, we use the full BN descriptor set (4946 observations, 24 variables) and register 8 clusters to train the EPX model:

```
cl <- parallel::makeCluster(8)
doParallel::registerDoParallel(cl)

set.seed(761)
BN <- rbind(BNsample, BNhold)
model <- epx(x = BN[,-25], y = BN[,25], classifier.args = list(ntree = 150),
                    computing = "parallel")

parallel::stopCluster(cl)
```

We indicate to `epx` that we wish to compute in parallel via the `computing` argument in line 136 – 137. Without specifying this argument, `epx` will override the clusters registered and compute sequentially.

When computing sequentially, all the relevant functions in **EPX** can be reproduced as long as a seed is set via the usual `set.seed` function. This also applies for parallel computing. However for reproducibility when in parallel, even if the same seed has been set, if the number of clusters has changed, the results of functions involving "randomness" (such as any of those involving a random forest) will change.

Due to the complexity of reproducing results when parallel computing is involved, despite a seed and registering the same number clusters, occasionally it may seem as though re-running certain functions such as `epx` produces different results. This is fixed by stopping the clusters as in line 139 and re-running lines 131 to 139 to re-register the desired clusters anew.

4. Further capabilities

4.1. Additional base classifiers

Aside from random forest via **randomForest** (Liaw and Wiener 2002), the other possible base classifiers allowed in the `classifier` argument include logistic regression and neural networks. Logistic regression is done using the `glm` function from the **stats** package (R Core Team 2020) (line 140). No additional arguments regarding logistic regression are allowed.

A single-hidden-layer neural network may also be specified as the base classifier and is done using the `nnet` function from the **nnet** package (Venables and Ripley 2002). With this base

classifier, we allow the number of units in the hidden layer to be set via `classifier.args` as demonstrated in lines 142 – 143.

```
140 model.log <- epx(x = BNsample[,-25], y = BNsample[,25], classifier = "logistic")
141
142 model.nn <- epx(x = BNsample[,-25], y = BNsample[,25], classifier = "neural",
143                 classifier.args = list(size = 2))
```

Other classifiers and their common arguments will be implemented in **EPX** as needed in future versions.

4.2. Additional performance measures

Apart from the default performance measure AHR, the three other measures that may be used are initial enhancement (IE), TOP1, and rank last (RKL). All four metrics are available in **EPX** as separate functions named `AHR`, `IE`, `TOP1`, and `RKL`, respectively. Each function requires a vector of predicted probabilities of relevance for the argument `phat`, and a binary vector indicating the true response for all observations for the argument `y`.

IE is defined in Tomal *et al.* (2015, 2016), and since IE is a rescaling of the precision given a cutoff, it leads to the same conclusion as AHR. Though IE is often reported in QSAR studies, unlike AHR which is bounded by 1, IE is unbounded (larger values are better). Note that IE is the only performance metric in **EPX** where the user may specify an additional argument via `performance.args` in the functions `epx`, `predict`, and `cv.epx`. Specifically, the user may set their desired shortlist cutoff for IE.

TOP1 is a performance measure that takes on a value of either 1 or 0. After sorting the observations by their predicted probabilities of relevance in decreasing order so the first ranked observation has the highest probability of relevance, if the first ranked observation is truly relevant, TOP1 has a value of 1. Otherwise TOP1 is 0. If there are ties for the first rank, all the corresponding observations must be truly relevant for TOP1 to score 1.

Again after ranking the observations as done for TOP1, RKL in contrast is the rank of the last truly relevant observation. Hence, RKL can take on integer values from 1 to n, where n is the total number of observations. If there are ties, the last object in the tied group determines RKL. That is, if all n objects are tied at the first rank but only one object is truly relevant, RKL will have a value of n.

Using the predictions attained in the Predict section on the samples of the BN descriptor set withheld from `BNsample` (used to train the model), we can calculate the four performance measures as follows:

```
144 > AHR(y = BNhold$y, phat = predshold)
145 [1] 0.2347868
146 > IE(y = BNhold$y, phat = predshold, cutoff = 50)
147 [1] 16.61474
148 > TOP1(y = BNhold$y, phat = predshold)
149 [1] 1
150 > RKL(y = BNhold$y, phat = predshold)
151 [1] 3946
```

5. Conclusion

Tomal *et al.* (2015, 2016) demonstrate how using an ensemble of phalanxes results in better performance in the area of QSAR studies. That is, we attain better ranking using distinct statistical models trained on disjoint subsets of variables than with the popular random forest, regularized random forest, balanced random forest and logistic regression model.

The **EPX** package demonstrates an implementation of the phalanx-formation algorithm and related functions. Currently, **EPX** is prepared for similar use-cases as those shown in Tomal *et al.* (2015, 2016, 2019) when the aim is binary classification with a rare class of objects and performance is judged via ranking. Future work may include the addition of other base classifiers and more notably, an adaptation for analogous regression problems.

Acknowledgments

The **EPX** package was funded by the Natural Sciences and Engineering Research Council of Canada and the Department of Statistics of the University of British Columbia.

References

Analytics R, Weston S (2020). *foreach: Provides Foreach Looping Construct for R*. R package version 1.5.0, URL https://CRAN.R-project.org/package=foreach.

Burden FR (1989). "Molecular identification number for substructure searches." *Journal of Chemical Information and Computer Sciences*, **29**(3), 225–227. doi:10.1021/ci00063a011. https://pubs.acs.org/doi/pdf/10.1021/ci00063a011, URL https://pubs.acs.org/doi/abs/10.1021/ci00063a011.

Liaw A, Wiener M (2002). "Classification and Regression by randomForest." *R News*, **2**(3), 18–22. URL http://CRAN.R-project.org/doc/Rnews/.

R Core Team (2020). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Revolution Analytics, Weston S (2019). *doSNOW: Foreach Parallel Adaptor for the 'snow' Package*. R package version 1.0.15, URL https://CRAN.R-project.org/package=doSNOW.

Tomal J, Welch W, Zamar R (2017). "Discussion of random-projection ensemble classification by T. I. Cannings and R. J. Samworth." *Journal of the Royal Statistical Society: Series B*, **79**(4), 1024–1025. URL http://dx.doi.org/10.1111/rssb.12228.

Tomal JH, Welch WJ, Zamar RH (2015). "Ensembling Classification Models Based on Phalanxes of Variables with Applications in Drug Discovery." *The Annals of Applied Statistics*, **9**(1), 69–93. ISSN 1932-6157. doi:10.1214/14-AOAS778.

Tomal JH, Welch WJ, Zamar RH (2016). "Exploiting Multiple Descriptor Sets in QSAR Studies." *Journal of Chemical Information and Modeling*, **56**(3), 501–509. ISSN 1549-9596, 1549-960X. doi:10.1021/acs.jcim.5b00663.

Tomal JH, Welch WJ, Zamar RH (2019). "Ensembles of phalanxes across assessment metrics for robust ranking of homologous proteins." 1706.06971.

Venables WN, Ripley BD (2002). *Modern Applied Statistics with S*. Fourth edition. Springer, New York. ISBN 0-387-95457-0, URL http://www.stats.ox.ac.uk/pub/MASS4.

Wang M (2005). *Statistical Methods for High Throughput Screening Drug Discovery Data*. Ph.D. thesis, University of Waterloo. URL http://etd.uwaterloo.ca/etd/y32wang2005.pdf.

Affiliation:

(i) Grace G. Hsu
Alumni, Department of Statistics
University of British Columbia
3182 Earth Sciences Building, 2207 Main Mall
Vancouver, BC, Canada V6T 1Z4
E-mail: grace.hsu@alumni.ubc.ca

(ii) Jabed H. Tomal, PhD
Assistant Professor
Department of Mathematics and Statistics
Thompson Rivers University
805 TRU Way, Kamloops, BC, Canada V2C 0C8
E-mail: jtomal@tru.ca

(iii) William J. Welch, PhD
Professor, Department of Statistics
University of British Columbia
3182 Earth Sciences Building, 2207 Main Mall
Vancouver, BC, Canada V6T 1Z4
E-mail: will@stat.ubc.ca

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
MMMMMM YYYY, Volume VV, Issue II *Submitted:* yyyy-mm-dd
doi:10.18637/jss.v000.i00 *Accepted:* yyyy-mm-dd