Journal of Statistical Software

November 2014, Volume 61, Issue 10. *http://www.jstatsoft.org/*

rFerns: An Implementation of the Random Ferns Method for General-Purpose Machine Learning

Miron Bartosz Kursa
University of Warsaw

Abstract

Random ferns is a very simple yet powerful classification method originally introduced for specific computer vision tasks. In this paper, I show that this algorithm may be considered as a constrained decision tree ensemble and use this interpretation to introduce a series of modifications which enable the use of random ferns in general machine learning problems. Moreover, I extend the method with an internal error approximation and an attribute importance measure based on corresponding features of the random forest algorithm. I also present the R package **rFerns** containing an efficient implementation of this modified version of random ferns.

Keywords: machine learning, random ferns, classification, R.

1. Introduction

Random ferns is a machine learning algorithm proposed by Özuysal, Fua, and Lepetit (2007) for matching the same elements between two images of the same scene, allowing one to recognize certain objects or trace them on videos. The original motivation behind this method was to create a simple and efficient algorithm by extending the naïve Bayes classifier; still the authors acknowledged its strong connection to decision tree ensembles like the random forest algorithm (Breiman 2001).

Since introduction, random ferns have been applied in numerous computer vision applications, like image recognition (Bosch, Zisserman, and Munoz 2007), action recognition (Oshin, Gilbert, Illingworth, and Bowden 2009) or augmented reality (Wagner, Reitmayr, Mulloni, Drummond, and Schmalstieg 2010). However, it has not gathered attention outside this field; thus, this work aims to bring this algorithm to a much wider spectrum of applications. In order to do that, I propose a generalized version of the algorithm, implemented in the R (R Core Team 2014) package **rFerns** (Kursa 2014) which is available from the Comprehensive R

Archive Network (CRAN) at http://CRAN.R-project.org/package=rFerns.

The paper is organized as follows. Section 2 briefly recalls the Bayesian derivation of the original version of random ferns, presents the decision tree ensemble interpretation of the algorithm and lists modifications leading to the **rFerns** variant. Next, in Section 3, I present the **rFerns** package and discuss the random ferns incarnations of two important features of the random forest: internal error approximation and attribute importance measure. Section 4 contains the assessment of **rFerns** using several well known machine learning problems. The results and computational performance of the algorithm are compared with the random forest implementation contained in the **randomForest** package (Liaw and Wiener 2002). The paper is concluded in Section 5.

2. Random ferns algorithm

Following the original derivation, let us consider a classification problem based on a dataset $(X_{i,j}, Y_i)$ with p binary attributes $X_{\cdot,j}$ and n objects $X_{i,\cdot}$ equally distributed over C disjoint classes (those assumptions will be relaxed in the later part of the paper). The generic *maximum a posteriori* (MAP) Bayes classifier classifies the object $X_{i,\cdot}$ as

$$Y_i^p = \arg\max_y \mathsf{P}(Y_i = y | X_{i,1}, X_{i,2}, \ldots, X_{i,p}); \tag{1}$$

according to the Bayes theorem, it is equal to

$$Y_i^p = \arg\max_y \mathsf{P}(X_{i,1}, X_{i,2}, \ldots, X_{i,p} | Y_i = y). \tag{2}$$

Although this formula is exact, it is not practically usable due to the huge (2^p) number of possible $X_{i,\cdot}$ value combinations, which most likely is much larger than the available number of training objects n thus making reliable estimation of the probability impossible.

The simplest solution to this problem is to assume complete independence of the attributes, what brings us to the naïve Bayes classification where

$$Y_i^p = \arg\max_y \prod_j \mathsf{P}(X_{i,j} | Y_i = y). \tag{3}$$

The original random ferns classifier (Özuysal *et al.* 2007) is an in-between solution defining a series of K random selections of D features $(\vec{j}_k \in \{1..P\}^D, k = 1, \ldots, K)$ treated using a corresponding series of simple exact classifiers (ferns), whose predictions are assumed independent and thus combined in a naïve way, i.e.,

$$Y_i^p = \arg\max_y \prod_k \mathsf{P}(X_{i,\vec{j}_k} | Y_i = y), \tag{4}$$

where $X_{i,\vec{j}_k}$ denotes $X_{i,j_k^1}, X_{i,j_k^2}, \ldots, X_{i,j_k^D}$. In this way one can still represent more complex interactions in the data, possibly achieving better accuracy than in the purely naïve case. On the other hand, such a defined fern is still very simple and manageable for a range of D values.

The training of the random ferns classifier is performed by estimating probabilities $\mathsf{P}(X_{i,\vec{j}_k} | Y_i = y)$ with empirical probabilities calculated from a training dataset $(X_{i,j}^t, Y_i^t)$ of size $n^t \times p$.

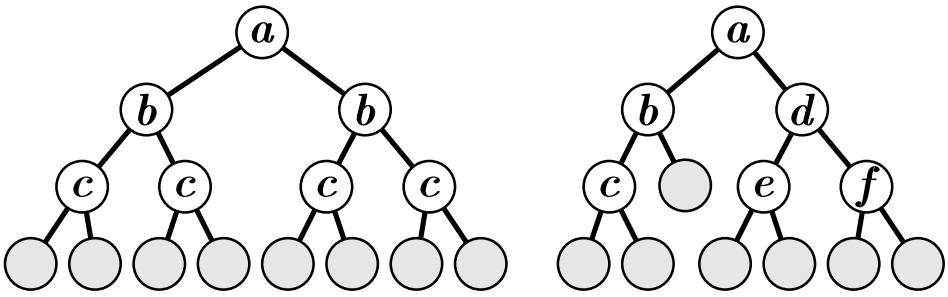

Figure 1: $D = 3$ fern shown in a form of a decision tree (left) and an example of a general decision tree (right). Consecutive letters mark different splitting criteria.

Namely, one uses frequencies of each class in each subspace of the attribute space defined by $\vec{j}_k$ assuming a Dirichlet prior, i.e.,

$$\widehat{\mathsf{P}}(X_{i,\vec{j}_k}|Y_i = y) = \frac{1}{\#L_{i,\vec{j}_k} + C}\left(1 + \#\left\{m \in L_{i,\vec{j}_k} : Y_m^t = y\right\}\right), \tag{5}$$

where # denotes the number of elements in a set and

$$L_{i,\vec{j}_k} = \left\{l \in \{1..n^t\} : \forall_{d\in\{1..D\}} X_{l,j_k^d}^t = X_{i,j_k^d}\right\} \tag{6}$$

is the set of training objects in the same leaf of fern k as object i.

2.1. Ensemble of decision trees interpretation

A fern implements a partition of the feature space into regions corresponding to all possible combinations of values of attributes $\vec{j}_k$. In this way it is equivalent to a binary decision tree of depth D for which all splitting criteria at tree level d are identical and split according to an attribute of index j^d, as shown in Figure 1. Consequently, because the attribute subsets $\vec{j}_k$ are generated randomly, the whole random ferns classifier is equivalent to a random subspace (Ho 1998) ensemble of K constrained decision trees.

Most ensemble classifiers combine predictions of their members through majority voting; this is also the case for random ferns when one considers scores $S_{i,\vec{j}_k}(y)$ defined as

$$S_{i,\vec{j}_k}(y) = \log \widehat{\mathsf{P}}(X_{i,\vec{j}_k}|Y_i = y) + \log C. \tag{7}$$

This mapping effectively converts the MAP rule into a majority voting

$$Y_i^p = \arg\max_y \sum_k S_{i,\vec{j}_k}(y). \tag{8}$$

The addition of $\log C$ causes that a fern that has no knowledge about the probability of the classes for some object will give it a vector of scores equal to zero.

2.2. Introduction of bagging

Using the ensemble of trees interpretation, in the **rFerns** implementation I was able to additionally combine the random subspace method with bagging, as it was shown to improve the

accuracy of similar ensemble classifiers (Friedman 2002; Breiman 2001; Panov and Džeroski 2007). This method restricts training of each fern to a *bag*, a collection of objects selected randomly by sampling with replacement n^t objects from an original training dataset, thus changing Equation 6 into

$$L_{i,\vec{j}_k} = \left\{ l \in B_k : \forall_{d \in \{1..D\}} X^t_{l,j^d_k} = X_{i,j^d_k} \right\}, \tag{9}$$

where B_k is a vector of indexes of the objects in the kth fern's bag.

In such a set-up, the probability that a certain object will not be included in a bag is $(1 - 1/n^t)^{n^t}$, thus each fern has a set of on average $n^t(1-1/n^t)^{n^t}$ ($n^t e^{-1} \approx 0.368 n^t$ for a large n^t) objects which were not used to build it. They form the *out-of-bag* (OOB) subsets which will be denoted here as B^*_k.

2.3. Generalization beyond binary attributes

As the original version of the random ferns algorithm was formulated for datasets containing only binary attributes, the **rFerns** implementation had to introduce a way to also cope with continuous and categorical ones. In the Bayesian classification view, this issue should be resolved by postulating and fitting some probability distribution over each attribute. However, this approach introduces additional assumptions and possible problems connected to the reliability of fitting.

In the decision tree ensemble view, each non-terminal tree node maps a certain attribute to a binary split using some criterion function, which is usually a greater-than comparison with some threshold value ξ in case of continuous attributes (i.e., $f_\xi : x \to (x > \xi)$) and a test whether it belongs to some subset of possible categories Ξ in case of categorical attributes (i.e., $f_\Xi : x \to (x \in \Xi)$).

In most *classification and regression trees* (CART) and CART-based algorithms (including random forest) the ξ and Ξ parameters of those functions are greedily optimized based on the training data to maximize the 'effectiveness' of the split, usually measured by the information gain in decision it provides. However, in order to retain the stochastic nature of random ferns the **rFerns** implementation generates them at random, similar to the Extra-trees algorithm by Geurts, Ernst, and Wehenkel (2006). Namely, when a continuous attribute is selected for creation of a fern level a threshold ξ is generated as a mean of two randomly selected values of it. Correspondingly, for a categorical attribute Ξ is set to a random one of all possible subsets of all categories of this attribute, except for the two containing respectively all and none of the categories.

Obviously, completely random generation of splits can be less effective than optimizing them in terms of the accuracy of a final classifier; the gains in computational efficiency may also be minor due to the fact that it does not change the complexity of the split building. However, in this way the classifier can escape certain overfitting scenarios and unveil more subtle interactions. This and the more even usage of attributes may be beneficial both for the robustness of the model and the accuracy of the importance measure it provides.

While in this generalization the scores depend on thresholds ξ and Ξ, from now on I will denote them as S_{i,F_k} where F_k contains $\vec{j}_k$ and necessary thresholds.

2.4. Unbalanced classes case

When the distribution of the classes in the training decision vector becomes less uniform, its contribution to the final predictions of a Bayes classifier increases, biasing learning towards the recognition of larger classes. Moreover, the imbalance may reach the point where it prevails the impact of attributes, making the whole classifier always vote for the largest class.

The original random ferns algorithm was developed under the assumption that the classes are of equal size, however such a case is very rare in a general machine learning setting and so the **rFerns** implementation has to cope with that problem as well. Thus, it is internally enforcing balance of class' impacts by dividing the counts of objects of a certain class in a current leaf by the fraction of objects of that class in the bag of the current fern – this is equivalent to a standard procedure of oversampling under-represented classes so that the number of objects of each class is equal within each bag.

Obviously there exist exceptional use cases when such a heuristic may be undesired, for instance when the cost of misclassification is not uniform. Then, this procedure might be reversed or replaced with another prior by modifying the raw scores before the voting is applied.

3. The rFerns package

The training of a random ferns model is performed by the `rFerns` function; it requires two parameters, the number of ferns K and the depth of each one D, which should be passed via the `ferns` and `depth` arguments respectively. If not given, $K = 1000$ and $D = 5$ are assumed. The current version of the package supports depths in the range 1..15. The training set can be given either explicitly by passing the predictor data frame and the decision vector, or via the usual formula interface:

```
R> model <- rFerns(Species ~ ., data = iris, ferns = 1000, depth = 5)
R> model <- rFerns(iris[, -5], iris[, 5])
```

The result is an S3 object of a class '`rFerns`', containing the ferns' structures F_k and fitted scores' vectors for all leaves.

To classify new data, one should use the `predict` method of the '`rFerns`' class. It will pull the dataset down each fern, assign each object the score vector from the leaf it ended in, sum the scores over the ensemble and find the predicted classes.

For instance, let us set aside the even objects of the `iris` data as a test set and train the model on the rest:

```
R> trainSet <- iris[c(TRUE, FALSE), ]
R> testSet <- iris[c(FALSE, TRUE), ]
R> model <- rFerns(Species ~ ., data = trainSet)
```

Then, the confusion matrix of predictions on a test set can be obtained by:

```
R> table(Prediction = predict(model, testSet), Truth = testSet[["Species"]])
```

```
            Truth
Prediction   setosa versicolor virginica
  setosa         25          0         0
  versicolor      0         24         1
  virginica       0          1        24
```

Adding `scores = TRUE` to the `predict` call makes it return raw class scores. The following code will extract scores of the first three objects of each class in the test set:

```
R> testScores <- predict(model, testSet, scores = TRUE)
R> cbind(testScores, trueClass = testSet[["Species"]])[c(1:3, 26:28,
+     51:53), ]
```

```
       setosa versicolor  virginica  trueClass
1   0.6083220 -0.8455781 -1.5057431     setosa
2   0.6672012 -0.9395135 -1.5468613     setosa
3   0.6255577 -0.9055999 -1.5120577     setosa
26 -1.3485694  0.2708711 -0.2595879 versicolor
27 -1.0709297  0.5555988 -0.7916823 versicolor
28 -1.2142952  0.4807975 -0.6461870 versicolor
51 -1.5922738 -0.1043981  0.3300516  virginica
52 -1.7083186 -0.2710267  0.4467801  virginica
53 -1.6010488 -0.6833165  0.6618151  virginica
```

3.1. Error estimate

By design, machine learning methods usually produce a highly biased result when tested on the training data; to this end, one needs to perform external validation to reliably assess its accuracy. However, in a bagging ensemble we can perform a sort of internal cross-validation where for each training set object prediction is built by having only those of the base classifiers vote which did not use this object for their training, i.e., which had it in their OOB subsets. This idea has been originally used in the random forest algorithm, and can be trivially transferred to any bagging ensemble, including the **rFerns** version of random ferns. In this case the OOB predictions Y_i^* will be given by

$$Y_i^* = \arg\max_y \sum_{k:i \in B_k^*} S_{i,F_k}(y) \tag{10}$$

and can be compared with the true classes Y_i to calculate the OOB approximation of the overall error.

On the R level, OOB predictions are always calculated when training an `rFerns` model; when its corresponding object is printed, the overall OOB error and confusion matrix are shown, along with the training parameters:

```
R> print(model)
```

```
 Forest of 1000 ferns of a depth 5.

 OOB error 5.33%; OOB confusion matrix:
            True
Predicted    setosa versicolor virginica
  setosa         25          0         0
  versicolor      0         24         3
  virginica       0          1        22
```

One can also access raw OOB predictions and scores by executing the `predict` method without providing new data to be classified:

```
R> oobPreds <- predict(model)
R> oobScores <- predict(model, scores = TRUE)
R> cbind(oobScores, oobClass = oobPreds, trueClass = trainSet$Species)[c(1:3,
+     26:28, 51:53), ]

      setosa versicolor  virginica   oobClass  trueClass
1   277.9060 -407.55145 -602.21266     setosa     setosa
2   268.6165 -386.42968 -542.61370     setosa     setosa
3   304.4445 -451.73002 -648.26384     setosa     setosa
26 -425.2157   63.64338  -71.56243 versicolor versicolor
27 -573.6673   45.77139   24.85006 versicolor versicolor
28 -553.0338  113.32022  -49.42665 versicolor versicolor
51 -451.9256 -231.65799  207.49580  virginica  virginica
52 -571.6487 -222.92446  234.47927  virginica  virginica
53 -589.4160 -208.51861  229.84709  virginica  virginica
```

Note that for a very small value of K some objects may manage to appear in every bag and thus get an undefined OOB prediction.

3.2. Importance measure

In addition to the error approximation, the random forest also uses the OOB objects to calculate the attribute importance. It is defined as the difference in the accuracy on the original OOB subset and the OOB subset with the values of a certain attribute permuted, averaged over all trees in the ensemble.

Such a measure can also be grafted on any bagging ensemble, including **rFerns**; moreover, one can make use of scores and replace the difference in accuracy with the mean difference of the scores of the correct class, in this way extracting importance information even from the OOB objects that are misclassified. Precisely, such a defined random ferns importance of an attribute a equals

$$I_a = \frac{1}{\#A(a)} \sum_{k \in A(a)} \frac{1}{\#B_k^*} \sum_{i \in B_k^*} \left(S_{i,F_k}(Y_i) - S^p_{i,F_k}(Y_i) \right), \tag{11}$$

where $A(a) = \{k : a \in \vec{j}_k\}$ is a set of ferns that use attribute a and S^p_{i,F_k} is S_{i,F_k} estimated on a permuted X^t in which the values of attribute a have been shuffled.

One should also note that the fully stochastic nature of selecting attributes for building individual ferns guarantees that the attribute space is evenly sampled and thus all, even marginally relevant attributes are included in the model for a large enough ensemble.

Calculation of the variable importance can be triggered by adding `importance = TRUE` to the call to `rFerns`; then, the necessary calculations will be performed during the training process and the obtained importance scores placed into the `importance` element of the '`rFerns`' object.

```
R> model <- rFerns(Species ~ ., data = iris, importance = TRUE)
R> model[["importance"]]
```

```
             MeanScoreLoss SdScoreLoss
Sepal.Length     0.1748790 0.006270534
Sepal.Width      0.1578244 0.005121205
Petal.Length     0.3195912 0.010456676
Petal.Width      0.2796645 0.010555186
```

4. Assessment

I tested **rFerns** on 7 classification problems from the R's **mlbench** (Leisch and Dimitriadou 2012) package, namely DNA (DNA), Ionosphere (ION), Pima Indian Diabetes (PIM), Satellite (SAT), Sonar (SON), Vehicle (VEH) and Vowel (VOW).

4.1. Accuracy

For each of the testing datasets, I built 10 random ferns models for each of the depths in range $\{1..15\}$ and number of ferns equal to 5000 and collected the OOB error approximations.

Next, I used these results to find optimal depths for each dataset (D_b) – for simplicity I selected the value for which the mean OOB error from all iterations was minimal.

Finally, I verified the error approximation by running 10-fold stochastic cross-validation. Namely, the set was randomly split into test and training subsets, composed respectively of 10% and 90% of objects; the classifier was then trained on a training subset and its performance was assessed using the test set. Such a procedure has been repeated ten times.

For comparison, I also built and cross-validated 10 random forest models with 5000 trees. The ensemble size was selected so that both algorithms would manage to converge for all problems.

The results of these tests are summarized in Table 1. One can see that as in the case of the random forest, the OOB error approximation is a good estimate of the final classifier error. It also serves well as an optimization target for the fern depth selection – only in case of the Sonar data the naïve selection of the depth giving minimal OOB error led to a suboptimal final classifier, however one should note that the minimum was not significantly different in this case.

Based on the OOB approximations, the forest outperforms ferns in all but one case; yet the results of the cross-validation show that those differences are in practice masked by the

	Dataset	DNA	ION	PIM	SAT
	Dataset size	3186×180	351×34	392×8	6435×36
OOB [%]	Ferns 5	6.03 ± 0.18	7.32 ± 0.23	24.69 ± 0.48	18.40 ± 0.13
	Ferns 10	6.56 ± 0.11	7.35 ± 0.22	27.93 ± 0.30	15.46 ± 0.06
	Ferns D_b	6.03 ± 0.18	7.07 ± 0.40	23.95 ± 0.31	14.33 ± 0.05
	Forest	4.13 ± 0.09	6.55 ± 0.00	21.76 ± 0.36	7.87 ± 0.06
CV [%]	Ferns 5	6.52 ± 1.66	7.78 ± 3.41	24.50 ± 6.75	18.60 ± 1.32
	Ferns 10	6.96 ± 1.30	8.61 ± 3.81	29.50 ± 6.10	15.92 ± 1.30
	Ferns D_b	5.92 ± 1.41	5.00 ± 3.88	24.00 ± 6.99	14.32 ± 0.88
	Forest	4.20 ± 0.99	6.11 ± 4.68	21.00 ± 3.94	7.75 ± 1.54
	D_b	5	3	7	15

	Dataset	SON	VEH	VOW
	Dataset size	208×60	846×18	990×10
OOB [%]	Ferns 5	19.71 ± 0.60	31.17 ± 0.49	13.70 ± 0.52
	Ferns 10	14.18 ± 1.12	29.52 ± 0.23	4.42 ± 0.26
	Ferns D_b	13.13 ± 0.64	28.83 ± 0.49	2.41 ± 0.19
	Forest	15.38 ± 0.64	25.48 ± 0.18	2.13 ± 0.11
CV [%]	Ferns 5	22.38 ± 6.37	32.94 ± 4.15	17.07 ± 3.10
	Ferns 10	14.29 ± 5.94	29.41 ± 7.48	5.25 ± 1.64
	Ferns D_b	18.10 ± 4.92	28.71 ± 5.69	2.22 ± 1.77
	Forest	19.52 ± 8.53	22.35 ± 4.33	2.22 ± 1.70
	D_b	12	15	15

Table 1: OOB and cross-validation error of the random ferns classifier for 5000 ferns of a depth equal to 5, 10 and optimal over $\{1..15\}$, D_b. These results are compared to the accuracy of a random forest classifier composed of 5000 trees. Prediction errors are given as a mean and standard deviation over 10 repetitions of training for OOB and 10 iterations for cross-validation.

natural variability of both classifiers. Only in case of the Satellite data the random forest clearly achieves almost a two times smaller error.

4.2. Importance

To test the importance measure, I used two datasets for which the importance of attributes should follow certain patterns.

Each object in the DNA dataset (Noordewier, Towell, and Shavlik 1991) represents a 60-residue DNA sequence in a way so that each consecutive triplet of attributes encodes one residue. Some of the sequences contain a boundary between exon and intron (or intron and exon[1]) regions of the sequence – the objective is to recognize and classify those sequences. All sequences were aligned in a way that the boundary always lies between the 30th and 31st residue; while the biological process of recognition is local, the most important attributes should be those describing residues in the vicinity of the boundary.

Objects in the Sonar dataset (Gorman and Sejnowski 1988) correspond to echoes of a sonar signal bounced off either a rock or a metal cylinder (a model of a mine). They are repre-

[1]The direction of a DNA sequence is significant, so these are separate classes.

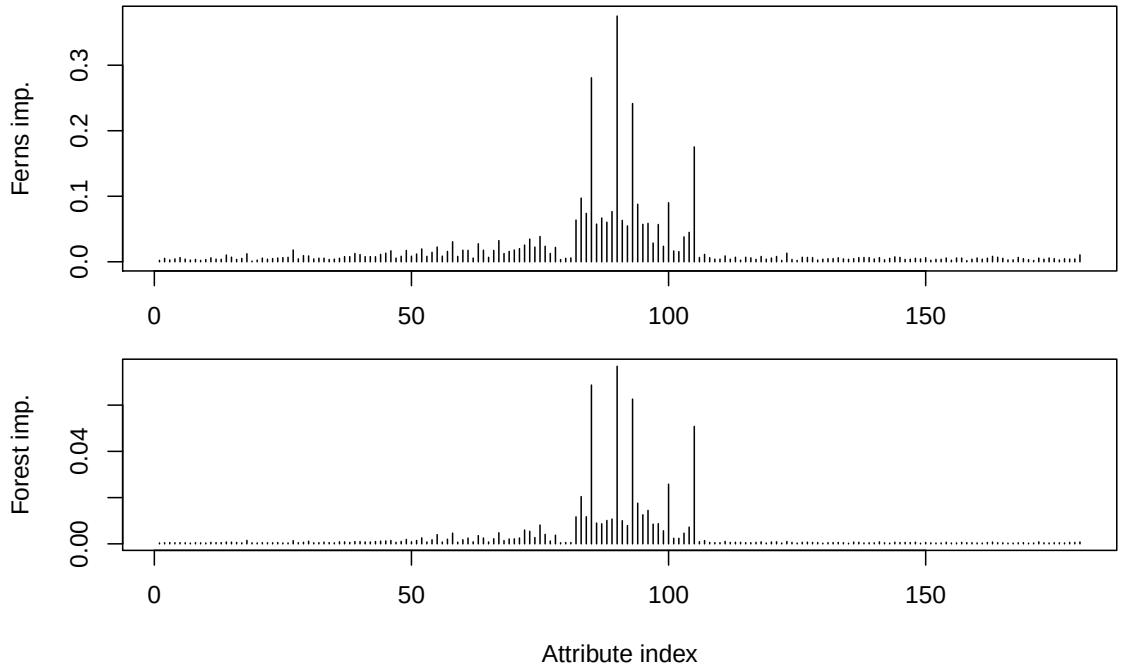

Figure 2: Attribute importance for the DNA data, generated by random ferns (top) and, for comparison, by random forest (bottom). Note that the importance peaks around the 90th attribute, corresponding to an actual splicing site.

	Dataset	Forest [s]	Ferns 10 [s]	Speedup	Ferns D_b [s]	Speedup	D_b
Just training	DNA	30.13	1.96	15.41	0.65	46.28	5
	ION	3.41	0.81	4.21	0.06	55.35	3
	PIM	2.45	0.97	2.53	0.24	10.29	7
	SAT	136.91	4.08	33.55	75.55	1.81	15
	SON	3.40	0.78	4.37	2.95	1.15	12
	VEH	8.00	1.67	4.80	46.04	0.17	15
	VOW	93.92	4.77	19.69	140.26	0.67	15
With importance	DNA	456.16	4.37	104.38	1.87	244.31	5
	ION	7.34	1.46	5.02	0.25	29.32	3
	PIM	3.53	1.06	3.34	0.35	10.17	7
	SAT	289.26	8.77	32.98	82.80	3.49	15
	SON	4.83	0.93	5.18	3.13	1.54	12
	VEH	17.26	2.22	7.77	47.00	0.37	15
	VOW	99.26	5.40	18.39	141.13	0.70	15

Table 2: Training times of the **rFerns** and **randomForest** models made for 5000 base classifiers, with and without importance calculation. Times are given as a mean over 10 repetitions.

sented as power spectra, thus each adjoining attribute value corresponds to the signal power contained within a consecutive frequency interval. In this way one may expect that there

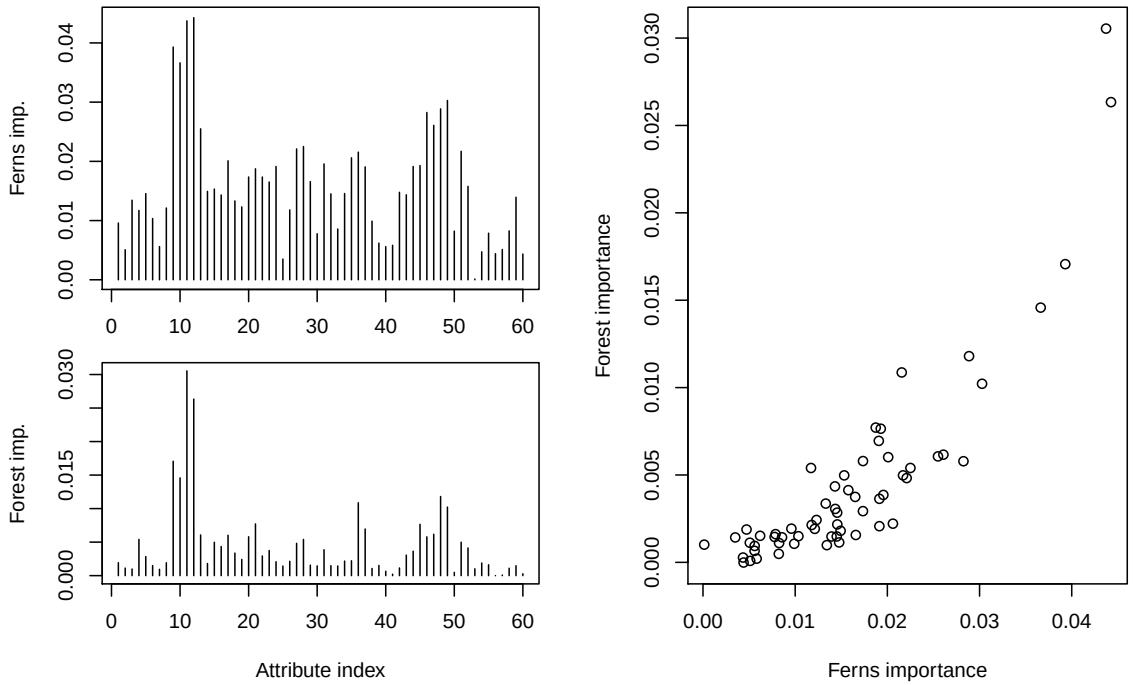

Figure 3: Importance measure for Sonar data, generated by random ferns (top right) and, for comparison, by random forest (bottom right). These two measures are compared in a scatter plot (left).

are frequency bands in which echoes significantly differ between classes, what would manifest itself as a set of peaks in the importance measure vector.

For both of these datasets, I calculated the importance measure using 1000 ferns of depth 10. As a baseline, I used the importance calculated using the random forest algorithm with 1000 trees.

The results are presented in Figures 2 and 3. The importance measures obtained in both cases are consistent with the expectations based on the datasets' structures – for DNA, one can notice a maximum around attributes 90–96, corresponding to the actual cleavage site location. For Sonar, the importance scores reveal a band structure which likely corresponds to the actual frequency intervals in which the echoes differ between stone and metal.

Both results are also qualitatively in agreement with those obtained from random forest models. Quantitative difference comes from the completely different formulations of both measures and possibly the higher sensitivity of ferns resulting from its fully stochastic construction.

4.3. Computational performance

In order to compare training times of the **rFerns** and **randomForest** implementations, I trained both models on all 7 benchmark datasets for 5000 ferns/trees, and, in case of ferns, depths 10 and D_b. Then I repeated this procedure, this time making both algorithms calculate the importance measure during training.

I repeated both tests 10 times to stabilize the results and collected the mean execution times; the results are summarized in Table 2. The results show that the usage of **rFerns** may result in significant speedups in certain applications; best speedups are achieved for the datasets with larger number of objects, which is caused by the fact that random ferns' training time scales linearly with the number of objects, while random forest's $\sim n \log n$.

Also the importance can be calculated significantly faster by **rFerns** than by **randomForest**, and the gain increases with the size of the dataset.

rFerns is least effective for datasets which require large depths of the fern – in case of the Vowel and Vehicle datasets it was even slower than random forest. However, one should note that while the complexity of random ferns $\sim 2^D$, its accuracy usually decreases much less dramatically when decreasing D from its optimal value – in this way one may expect an effective trade-off between speed and accuracy.

5. Conclusions

In this paper, I presented the package **rFerns**, a general-purpose implementation of the random ferns, a fast, ensemble-based classification method. Slight modifications of the original algorithm allowed me to additionally implement OOB error approximation and an attribute importance measure.

Presented benchmarks showed that such an algorithm can achieve accuracies comparable to the random forest algorithm while usually being much faster, especially for large datasets.

Also the importance measure proposed in this paper can be calculated very quickly and proved to behave in a desired way and be in agreement with the results of the random forest; however the in-depth assessment of its quality and usability for feature selection and similar problems requires further research.

Acknowledgments

This work has been financed by the National Science Centre, grant 2011/01/N/ST6/07035. Computations were performed at ICM, grant G48-6.

References

Bosch A, Zisserman A, Munoz X (2007). "Image Classification Using Random Forests and Ferns." In *Proceedings of the 11th International Conference on Computer Vision*, pp. 1–8.

Breiman L (2001). "Random Forests." *Machine Learning*, **45**(1), 5–32.

Friedman JH (2002). "Stochastic Gradient Boosting." *Computational Statistics & Data Analysis*, **38**(4), 367–378.

Geurts P, Ernst D, Wehenkel L (2006). "Extremely Randomized Trees." *Machine Learning*, **63**(1), 3–42.

Gorman PR, Sejnowski TJ (1988). "Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets." *Neural Networks*, **1**(1), 75–89.

Ho TK (1998). "The Random Subspace Method for Constructing Decision Forests." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **20**(8), 832–844.

Kursa MB (2014). ***rFerns**: Random Ferns Classifier*. R package version 1.0.0, URL http://CRAN.R-project.org/package=rFerns.

Leisch F, Dimitriadou E (2012). ***mlbench**: Machine Learning Benchmark Problems*. R package version 2.1-1, URL http://CRAN.R-project.org/package=mlbench.

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

Noordewier MO, Towell GG, Shavlik JW (1991). "Training Knowledge-Based Neural Networks to Recognize Genes." In RP Lippmann, JE Moody, DS Touretzky (eds.), *Advances in Neural Information Processing Systems*, volume 3, pp. 530–536. Morgan-Kaufmann.

Oshin O, Gilbert A, Illingworth J, Bowden R (2009). "Action Recognition Using Randomised Ferns." In *Proceedings of the IEEE 12th International Conference on Computer Vision Workshops (ICCV Workshops 2009)*, pp. 530–537.

Özuysal M, Fua P, Lepetit V (2007). "Fast Keypoint Recognition in Ten Lines of Code." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2007)*, pp. 1–8.

Panov P, Džeroski S (2007). "Combining Bagging and Random Subspaces to Create Better Ensembles." In *Proceedings of the 7th International Conference on Intelligent Data Analysis*, pp. 118–129. Springer-Verlag, Berlin.

R Core Team (2014). "R: A Language and Environment for Statistical Computing." URL http://www.R-project.org/.

Wagner D, Reitmayr G, Mulloni A, Drummond T, Schmalstieg D (2010). "Real-Time Detection and Tracking for Augmented Reality on Mobile Phones." *IEEE Transactions on Visualization and Computer Graphics*, **16**(3), 355–368.

Affiliation:

Miron Bartosz Kursa
Interdisciplinary Centre for Mathematical and Computational Modelling
University of Warsaw
Warsaw, Poland
E-mail: M.Kursa@icm.edu.pl

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 61, Issue 10 *Submitted:* 2012-03-01
November 2014 *Accepted:* 2014-06-06