Final Project

Online Random Forests (and where to find them)

By team «Overhit»

Mikhail Sidorenko, Andrey Demidov, Anton Myshak, Alexey Topolnitskiy

Table of Contents

Abstract

How to prevent reducing of quality of your model's predictions if the data distribution is changing over time? How to train your models if data doesn't fit into memory? Online learning approaches!

In this project we considered online learning approaches applied to random forests. In many applications random forests can be an algorithm of choice due to it natural parallelism and high resistance to overfitting. This makes online random forests an actual area of research.

We considered three modifications of vanilla random forest, which make it work in online mode and compared the results with two online random forest algorithms. We implemented from scratch all algorithms, namely, decision tree, vanilla random forest, online decision tree, online random forest and Mondrian forest in pure Python. All implemented algorithms and Jupyter notebooks with experiments are available in the GitHub repository: https://github.com/smikhai1/RF-online.

Our experiments shown the domination of Mondrian forests among other algorithms in terms of performance and time complexity.

1. Introduction

First of all, we did a little research on what the online and offline learning are what are their differences. Depending on the training and prediction modes, machine learning (ML) can be divided into following groups:

- Offline training + batch prediction: it's a common approach in scientific world of ML and at Kaggle, when you have a dataset (with historical data as a rule), train your model on part of it and make predictions to evaluate the model on test dataset. In production this approach can be used for example with simple scripts that start the prediction automatically using the data from database.
- Offline training + predictions on demand: this is an approach when your model is trained on the historic data once, and then makes predictions by request from users. The main disadvantage of this approach is that it doesn't take into account any changes in the distribution of data and it may end up in a poor model accuracy in the future.
- Online training + predictions on demand: this is a what's called online ML, the model is dynamic and is updated as the new data comes. The ideal version of this approach is automated ML when your model trained and even tuned automatically. Both frameworks may take into account changing in data distribution over time, models can be trained on a very large dataset because it isn't necessary to load the whole dataset in RAM once a time.

All of these approaches are presented in Figure 1.

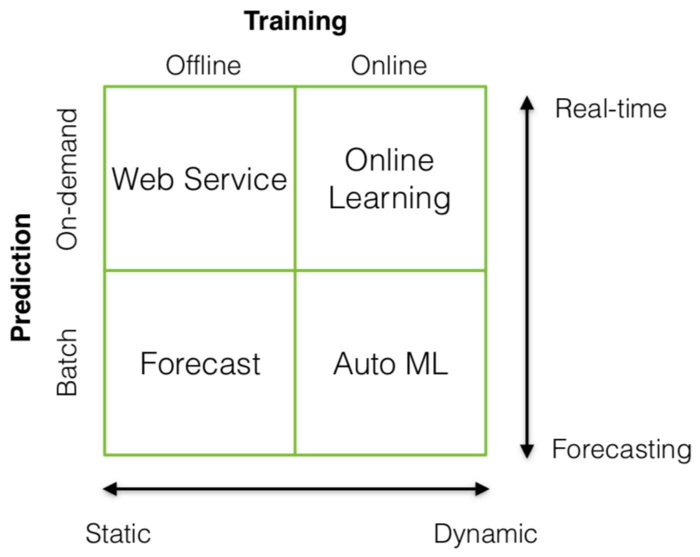


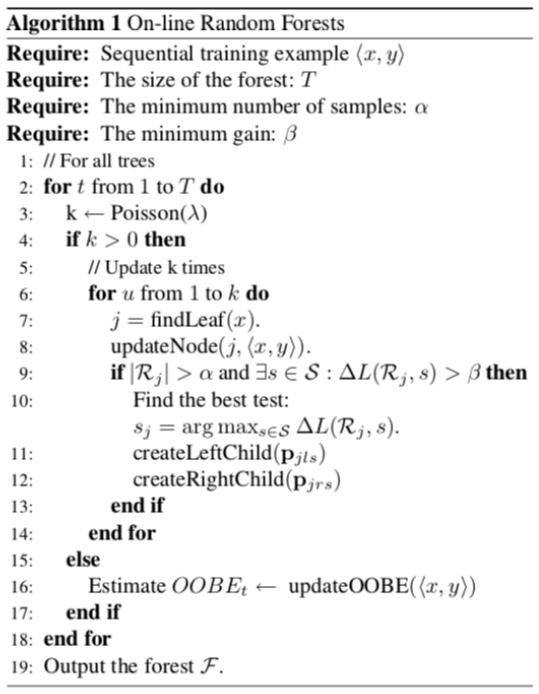
Figure 1. Types of machine learning in terms of training and prediction modes (source)

There are online learning frameworks, such as Vowpal Wabbit and Scikit-learn, which support online learning for the following ML algorithms: one-layer Neural Network, SVM, Naive Bayes, Ordinary Least Square, Ridge, Lasso and so on. But they don't support random forests, which in many problems can be an algorithm of choice due to it natural parallelism and high resistance to overfitting. This makes online random forests an actual area of research.

2. Review of related works

In our project we mainly concentrated on two works related to online random forests [Saffari et.al, 2009; Lakshminarayanan et.al, 2014].

Vanilla Random Forest, proposed in [Breiman, 2001], is a bagging over decision trees. Each tree builds on bootstrap from the initial dataset with random set of features at each split, using recursive procedure such as ID3, C4.5 or CART. The algorithm in its original form can be trained only on the full dataset, thus the natural question arises: how to adopt it for online mode, when each observation comes sequentially and doesn't store in memory.



Algorithm 1 On-line Random Forests

```
Require: Sequential training example $\langle x, y \rangle$
Require: The size of the forest: $T$
Require: The minimum number of samples: $\alpha$
Require: The minimum gain: $\beta$
// For all trees
for $t$ from 1 to $T$ do
    k ← Poisson($\lambda$)
    if $k > 0$ then
        // Update k times
        for $u$ from 1 to $k$ do
            $j$ = findLeaf($x$).
            updateNode($j$, $\langle x, y \rangle$).
            if $|\mathcal{R}_j| > \alpha$ and $\exists s \in \mathcal{S} : \Delta L(\mathcal{R}_j, s) > \beta$ then
                Find the best test:
                $s_j = \arg\max_{s \in \mathcal{S}} \Delta L(\mathcal{R}_j, s)$.
                createLeftChild($\mathbf{p}_{jls}$)
                createRightChild($\mathbf{p}_{jrs}$)
            end if
        end for
    else
        Estimate $OOBE_t$ ← updateOOBE($\langle x, y \rangle$)
    end if
end for
Output the forest $\mathcal{F}$.
```

Figure 2. Algorithm for training Online Random Forests (Saffari et.al, 2009)

In [Saffari et.al, 2009] to do bagging online they used modification of it from [Oza, 2001]. The idea is following: let's X is a random variable, number of occurrences of a particular sample in bootstrapped dataset, then probability for a sample to be presented in bootstrapped dataset k times is $P(X = k) = C_n^k p^k (1-p)^{n-k}$, where n is the size of bootstrapped dataset, p is a probability for an object to be extracted from the dataset. Then one can show that

$$P(X = k) = C_n^k p^k (1-p)^{n-k} = \frac{n!}{k!(n-k)!}(\frac{1}{n})^k (1 - \frac{1}{n})^{n-k} \to \frac{1}{k!}(1 - \frac{1}{n})^{n-k} \to \frac{exp(-1)}{k!},$$

when $n \to \infty$, i.e. for big datasets we can assume that the number of occurrences of each object in the bootstrapped dataset is drawn from Poisson(1). Therefore, the online bagging can be implemented by fitting each new observation to each tree in ensemble k times, where $k \sim Poisson(1)$.

To fit tree on a single observation they used so called evolving decision tree. It's a non-recursive procedure for building decision trees in which a node is split only after it saw a sufficient amount of data to make statistically significant split, α. Each node before splitting collects the statistics of observations come to it and then find the best predicate for splitting in terms of information gain among random subset of features and random thresholds (like in extremely randomized forest).

To account for changes in data distribution trees with low out-of-bag (OOB) scores can be randomly removed from the ensemble. OOB scores are calculated on trees for which $k = 0$.

The algorithm is shown in Figure 2.

In [Lakshminarayanan et.al, 2014] they stated the following. The primary difficulty with building online decision trees is their recursive structure. Data encountered once a split has been made cannot be used to correct earlier decisions. The quality of each split is estimated online as data arrive in the leaf, but since the entire training set is not available these quality measures are only estimates.

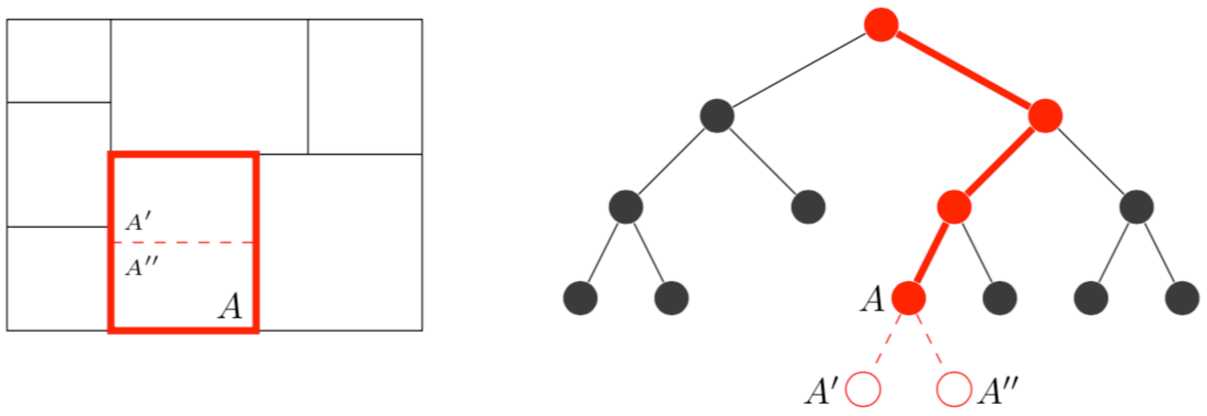


Figure 3. Mondrian decision tree [Lakshminarayanan et.al, 2014]

At any time, the online forest can be used to make predictions for unlabeled data points using the model built from the labelled data it has seen so far. To make a prediction for a query point x at time t, each tree computes, for each class k,

$$\eta_t^k = \frac{1}{N^e(A_t(x))} \sum_{\substack{(X_\tau, Y_\tau) \in A_t(x) \\ I_\tau = e}} \mathbb{I}\{Y_\tau = k\},$$

where $A_t(x)$ denotes the leaf of the tree containing x at time t, and $N^e(A_t(x))$ is the number of estimation points which have been counted in $A_t(x)$ during its lifetime. Similarly, the sum is over the labels of these points. The tree prediction is then the class which maximizes this value:

$$g_t(x) = argmax_k\{\eta_t^k(x)\}$$

The forest predicts the class which receives the most votes from the individual trees. Figure 3 shows what such tree is look like.

In Figure 4 the comparison of Mondrian tree and simple decision tree in $[0,1]^2$, wherex_1 and x_2 denote horizontal and vertical axis is provided.

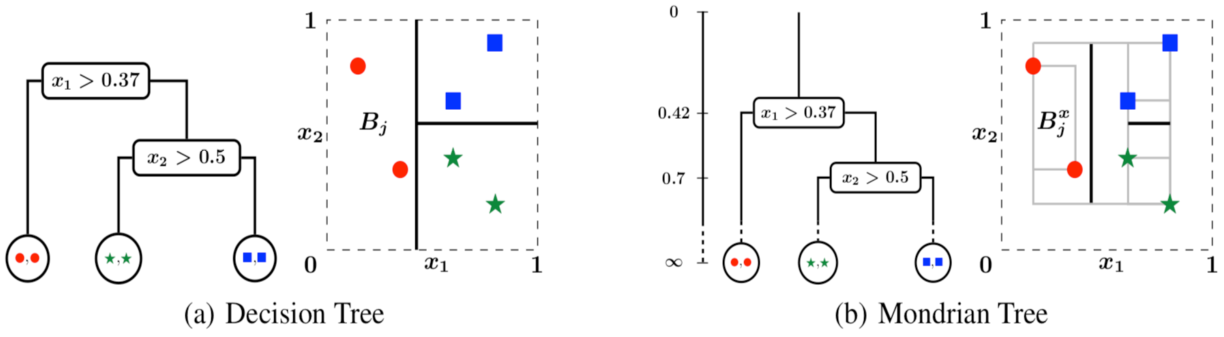


Figure 4. Comparison of Mondrian tree and vanilla decision tree.

Mondrian tree is embedded on a vertical time axis and each node associated with a time of split and the splits are committed only within the range of the training data in each block (denoted by gray rectangles). Let j denote the left child of the root: $B_j = (0, 0.37] \times (0, 1]$ denotes the block associated with red circles and $B_j^x \subseteq B_j$ is the smallest rectangle enclosing the two data points.

3. Description of data

For testing our implementations and checking appropriate quality we used three different datasets. First is Handwritten Digits USPS dataset. It is created for multiclass classification. It contains 7291 train and 2007 test image; each image is 16*16 grayscale pixels.

Second dataset is Letter Recognition Data Set. It consists of 20000 instances and 16 numerical attributes, and main objective of the dataset is to identify 26 different capital letters in the alphabet.

Finally, poisson mushrooms recognition dataset, which consists of characteristics of different specious of mushrooms. [Binary classification problem with 5686 train and 2438 test examples, which includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms.

4. Experimental setup

We consider classification problems.

In these experiments we firstly use offline implementations of Random Forest and Decision Tree. To make them work in online mode all train sets are divided into T batches and then online refitting of RF is modeled using three different strategies:

1. Initially dataset is presented as one batch. Then all batches are sequentially added to the dataset. Random Forest model is firstly trained on the initial dataset, and then completely refitted on the updated dataset as new batches added. After each refitting the model is evaluated on the test dataset. This process is repeated 10 times to account for the randomness in RF algorithm.
2. At each updating of the dataset, some number of trees in the ensemble is randomly removed (the less OOB score a tree has, the higher the probability of it to be removed) and then this number of tree is fit on the new data, OOB scores are updated for all new trees on the new data.
3. This strategy is pretty similar to the second one and models the situation when the data doesn't fit into RAM. The difference between this strategy and the second one is that instead of accumulating of the data we preserve the size of the dataset constant, using rolling-window, and refit the ensemble as in the second strategy but only on this dataset.

Due to the fact that all datasets are balanced, more or less, we use accuracy as quality metric.

After that, we consider different Random Forest algorithms, which are specially designed for serving in the online mode.

As the outcome of each experiment we want to obtain the dependency between the dataset size and the performance of the model for each dataset and for three models: offline DT, offline RF and online RF. Also we compare offline and online learning algorithms in time complexity.

5. Results

In Figure 5 the estimated performance vs. proportion of data of all tested algorithms are presented for three datasets. And in Table 1 time complexities of all algorithms are presented.

From the obtained results we can see that algorithms specially designed for online mode perform better than the adoptations of offline random forest for online mode in both accuracy and time complexity. One also can mention that Mondrian forest have some strange outliers in accuracy for Mushrooms dataset. It can be explained by the fact that this dataset is simple, i.e. all objects can be classified well using simple decisions rules as was stated in the corresponding UCI repository.

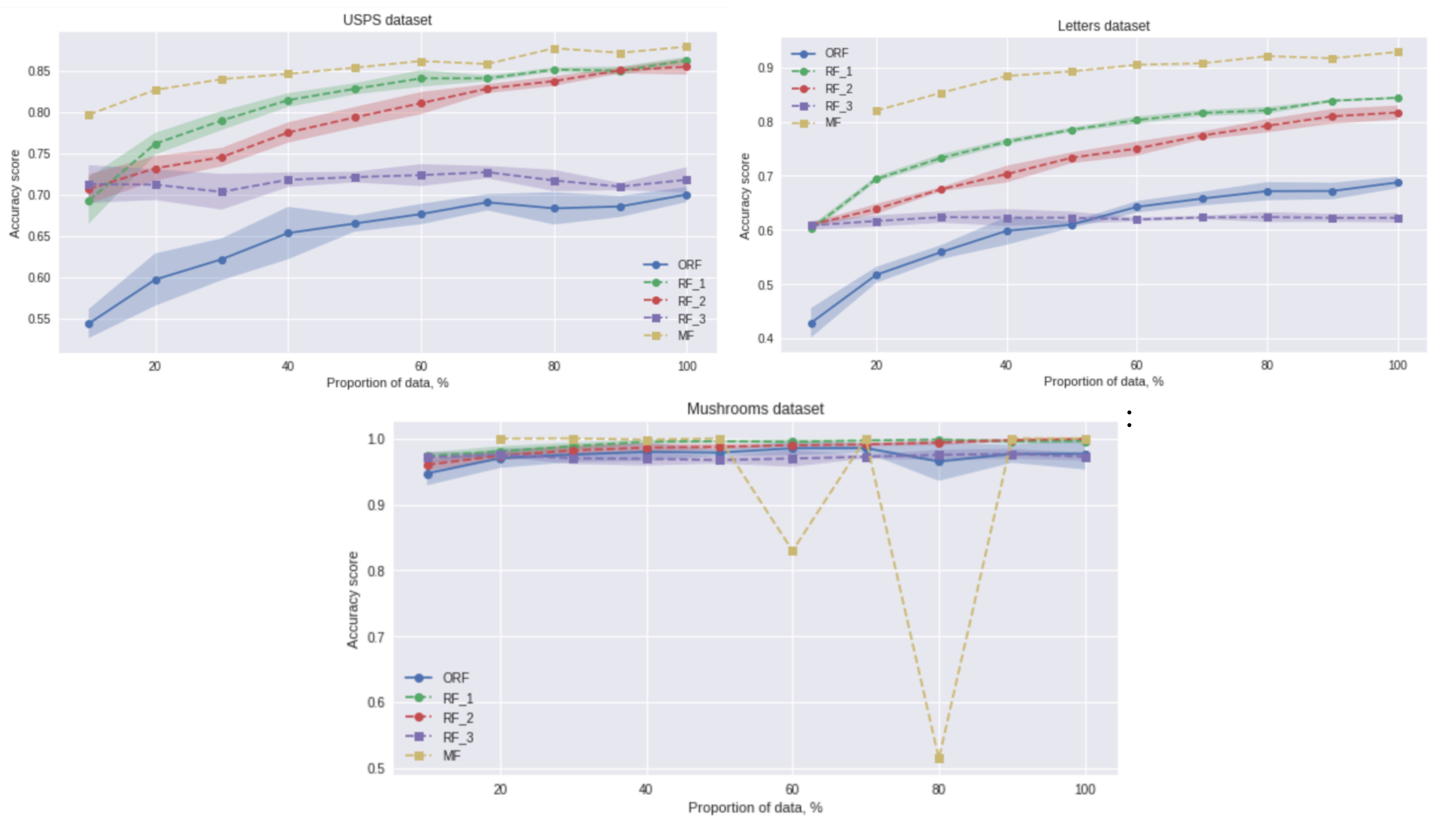


Figure 5. Accuracy vs Proportion of data plots for USPS, Letters and Mushrooms datasets.

Table 1. Time complexities of algorithms.

	RF-1	RF-2	RF-3	ORF	MRF
USPS	636.11 s	204.36 s	42.53 s	66.6 s	14.1 s
Letters	307.79 s	111.61 s	34.9 s	31.71 s	24.65 s
Mushrooms	32.87 s	14.66 s	9.0 s	3.3 s	2.61 s

Challenges

- Python implementations of vanilla algorithms work very slow
- Difficulties in understanding what online learning is because there are a lot of different views on it

Team contribution

Mikhail Sidorenko, team lead: implementation of online random forest from [Saffari, et. al], experiments with adaptations of vanilla random forests to online mode, report, presentation;

Andrey Demidov: implementation of Mondrian forest algorithms, optimization of vanilla algorithms, results evaluation, presentation;
Anton Myshak: implementation of adaptations for vanilla random forests, report, presentation;
Alexey Topolnitsky: implementation of vanilla algorithms, report, presentation.

Github repository:

https://github.com/smikhai1/RF-online

References

1. L. Breiman. Random forests. Machine Learning, 45(1): 5–32, 2001.
2. N. Oza and S. Russel. Online Bagging and Boosting. In Artificial Intelligence and Statistics, volume 3, 2001.
3. Saffari, C. Leistner, J. Santner, M. Godec, and H. Bischof. On-line random forests. In International Conference on Computer Vision Workshops (ICCV Workshops), pp. 1393–1400. IEEE, 2009.
4. B. Lakshminarayanan, Daniel M. Roy, Yee Whye Teh. Mondrian Forests: Efficient Online Random Forests.
5. https://www.kaggle.com/bistaumanga/usps-dataset
6. https://archive.ics.uci.edu/ml/datasets/letter+recognition
7. https://archive.ics.uci.edu/ml/datasets/mushroom