Learning on a Stream of Features with Random Forest

Jan Motl, Pavel Kordík

Czech Technical University in Prague,
Thákurova 9, 160 00 Praha 6, Czech Republic,
jan.motl@fit.cvut.cz, pavel.kordik@fit.cvut.cz

Abstract: We study an interesting and challenging problem, supervised learning on a stream of features, in which the size of the feature set is unknown, and not all features are available for learning while leaving the number of observations constant. In this problem, the features arrive one at a time, and the learner's task is to train a model equivalent to a model trained from "scratch". When a new feature is inserted into the training set, a new set of trees is trained and added into the current forest. However, it is desirable to correct the selection bias: older features has more opportunities to get selected into trees than the new features. We combat the selection bias by adjusting the feature selection distribution. However, while this correction improves accuracy of the random forest, it may require training of many new trees. In order to keep the count of the new trees small, we furthermore put more weight on more recent trees than on the old trees.

Keywords: random forest, incremental learning, online learning, sequential learning, stream learning

Problem formulation One of the common issues in machine learning is changing data and the need to keep the machine learning models up to date with the changing data. One of the successful simplifications is to assume that over time we are getting new samples. However, this article is concerned with the orthogonal problem — fast updates of models when new features arrive (see Figure 1).

Motivation Our original need for learning on a stream of features was due to our interest into propositionalization [3]. Propositionalization is a data preprocessing step, which converts relational data into a single data. And one of the persistent problems of propositionalization is that it generates a wast quantity of redundant and/or unpredictive features (e.g.: [3, 2]). Would not it be interesting to intelligently guide the propositionalization in order to avoid wasteful generation of these irrelevant features? Our previous research [4] answered this question positively — based on *univariate feature selection* on a stream of features, we obtained 10-fold acceleration of the propositionalization (while maintaining the accuracy of the downstream model

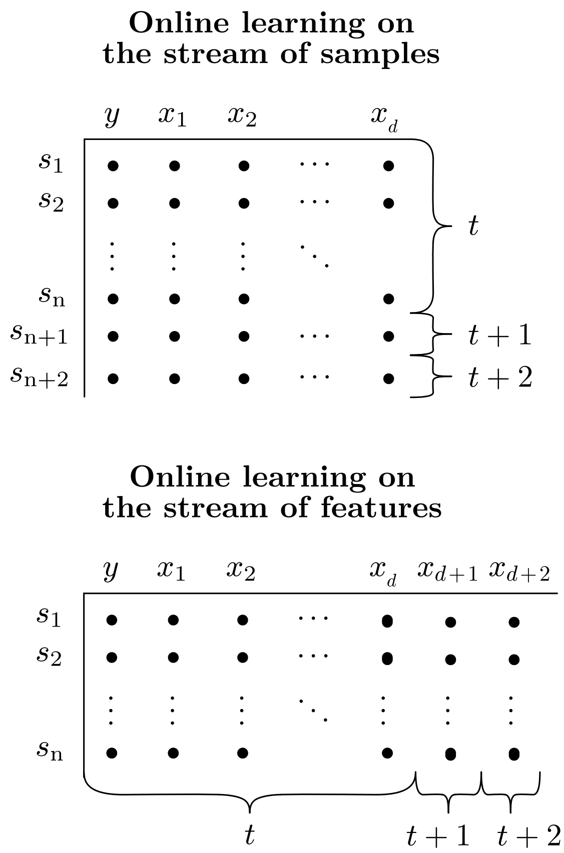

Figure 1: The difference between learning from a stream of samples (top) and a stream of features (bottom). In both cases, we have n samples and d features at time t. But at time $t+1$, we either have one more sample (top) or one more feature (bottom).

comparable to accuracy obtained on exhaustive propositionalization). However, our former research had evident weakness: it neglected to take into account possible interactions between features. This paper attempts to address this weakness.

Why not a feature selection filter? Features that are currently unpredictive may become predictive, as new features appear. For example, consider XOR problem, in which the binary label is determined by two binary features f_1 and f_2: $y = xor(f_1, f_2)$. Features f_1 and f_2 are individually unpredictive. But together, they define the label. Univariate feature selection filters (e.g.: based on information gain ratio) cannot correctly identify the change or the first feature relevance as the second feature is added in XOR problem. But models capable of modeling feature interactions (like random forests) can eventually identify these features as important.

Application Beside propositionalization, learning on a stream of features has another interesting use-case: Kaggle competitions. In these challenges, competitors are given a dataset and the team with the best model wins[1]. Based on the analysis of solutions of the past winners[2], one of the common differentiating factors is extensive feature engineering. However, competitive feature engineering is generally not a one-time task but rather an iterative process:

1. formulate a hypothesis (e.g.: log transformation of features will improve the accuracy of the downstream model),
2. test the hypothesis (e.g.: evaluate the change of accuracy of the downstream model),

where the choice of the next round of hypotheses is influenced based on the success of the previously evaluated hypotheses. Traditionally, the evaluation of the hypothesis required retraining of the model from scratch. Our solution is to update the current model. The benefit is evident: the update of the current model takes less time than retraining the model from scratch. And consequently, that gives us the freedom to test more hypotheses.

Random forest We take random forest [1] as a starting model to expand into an online implementation because it can deal with dirty data (e.g.: missing values, outliers, mix of numerical and nominal attributes,...) and given an implementation of a decision tree, it is easy to implement and reason about.

The key idea behind random forest classifier is that we make an ensemble of decision trees. In order to create diversity between the trees, it employs two strategies: bagging and random feature selection. Bagging is based on a random sampling of training instances with repetition. While random feature selection is without repetition. The count of features to select is one of the most tunable parameters of random forests [5] and multiple heuristics for the optimal value were provided in the literature. For simplicity of the following analysis, we assume that the count of the selected features is a fixed ratio of the count of all the features. We call the ratio *mtry*.

[1] See a list of all possible challenges at `https://www.kaggle.com/competitions`

[2] `http://blog.kaggle.com/category/winners-interviews/`

1 Implementation

Bias Whenever a new feature x_{new} arrives, we may train n new trees. And add the newly trained trees into the current random forest. Unfortunately, with this approach, the new features would be underrepresented in the forest in comparison to old features simply because the *old features had multiple opportunities* to get used in a tree while the *new feature had only one opportunity* to get used in a tree.

Consequently, earlier features would have a bigger impact (weight) in the forest than the newer features. This presents a bias, which is generally undesirable.

Variable count of trees The first intuitive improvement is to make sure that the new feature is actually always passed to the new trees (instead of leaving it on the chance). And instead of generating an arbitrary count of the trees, we can calculate the optimal count n that minimizes the random feature selection bias.

First, we introduce the notation. Let c be the count of how many times a feature x was passed to decision trees. And let *old* subscript describe some old feature and *new* subscript to describe the new feature. If we want to avoid the random feature selection bias, following should hold:

$$c_{new} = c_{old}. \tag{1}$$

Since

$$c_{new} = n \tag{2}$$

because the new feature is always selected and

$$c_{old} = mtry \cdot d_{old} + mtry \cdot n, \tag{3}$$

where d_{old} is the count of the old features, we get:

$$n = mtry \cdot d_{old} + mtry \cdot n. \tag{4}$$

Hence, we get the optimal n with:

$$n = \frac{mtry \cdot d_{old}}{1 - mtry}. \tag{5}$$

The issue with this approach is that if we keep adding d features one-by-one, the total count of the trees in the ensemble grows quadratically.

Tree weighting If we want to avoid the quadratic growth of the random forest, we may weight the late trees more than the former trees. While we could have calculated the tree weight analytically, we provide an algorithmic solution in Algorithm 1. In praxis, the advantage of the algorithmic solution is that it is self-correcting — if some of the assumptions are not fully fulfilled (e.g.: When we have 11 features and the feature selection ratio is 0.5, we can either

Algorithm 1: Random forest update, when a new feature arrives. Function `featureCnt()` returns count of features to sample.

```
Input: X: training data, y: training label, col: index of the new feature, treeCnt: cnt of trees to train,
       weightedFeatureUseCnt: bookkeeping vector initialized to zeros, ensemble: collection of trees.
Output: ensemble, treeWeight, weightedFeatureUseCnt.
featureUseCnt = zeros (col);
for i=1:treeCnt do
    oldFeatures = choice (1:col-1, featureCnt (col-1), replacement=False);
    features = [oldFeatures, col];
    samples = choice (nrow (x), nrow (x), replacement=True);
    tree = fitTree (X[samples, features], y[samples]);
    ensemble = [ensemble, tree];
    featureUseCnt[features]++ ;
end
treeWeight = avg (weightedFeatureUseCnt[1:col-1]) / (featureUseCnt[col] - avg (featureUseCnt[1:col-1]));
weightedFeatureUseCnt = weightedFeatureUseCnt + treeWeight*featureUseCnt;
```

select 5 or 6 features but not 5.5.), the error is not ignored (as it would be in a closed-form analytical solution) but is encoded in `weightedFeatureUseCount`. And each call of Algorithm 1 directly minimizes the error.

When scoring new samples, we evaluate trees in the ensemble and calculate the weighted average of the predictions (each generation of trees share the same treeWeight).

2 Experiments

We compare two online random forest implementations: **baseline** and **challenger**. In baseline, features are selected with uniform probability (like in ordinary random forest). In the challenger model, the new feature is always selected while the old features are selected with uniform probability[3]. Furthermore, we train an **offline** random forest with the same meta-parameters as the online random forest in order to depict the value of the online learning.

Protocol For each data set, we performed the following procedure 10 times: We randomly split the data set into training/testing subsets with stratified sampling with 2:1 ratio. Then we randomly permutate the feature order in the data set (because our proposal should work regardless of the feature ordering). Finally, on online random forests we perform incremental learning feature-by-feature (i.e.: first we train the random forest on the first feature, then we add the second feature into the forest,... and continue until the last feature is added into the forest). After adding the last feature, the final model is evaluated on the testing set with AUC (Area Under the Receiver Operating Characteristics). In the case of the offline random forest, we train the random forest just once on all the features.

Meta-parameters At each generation (addition of a new feature), we train 30 new trees. This value is recommended by Breiman [1] and we decided to go with it. For feature selection ratio, we used ⅔.

Data sets We used all 232 data sets (see Appendix A) from OpenML [6] that have a binary label (because we evaluate the models with AUC), less than 200000 samples (because of runtime) and less than 15 features (again, because of the runtime).

Results In 87% (201/232), the challenger model had higher average testing AUC than the baseline. Sign test on this statistic gives one-tail P-value $< 10^{-29}$. The average difference of the testing AUC across all the data sets was 2.10 percent point. Furthermore, in 71% (164/232), the challenger model had higher average testing AUC than the offline model (P-value $< 10^{-8}$). The table with the results and the code that generated the table is available from `https://github.com/janmotl/rf`.

3 Discussion

Overhead Challenger model, in comparison to baseline model, uses 3 more variables: `featureUseCount`, `weightedFeatureUseCount` and `treeWeight`. Each of these variables is (or fill in) a vector of length d, the count of features. Ignoring the differences in the data types, the total memory overhead is equivalent to 3 more training data

[3] This probability is smaller in the challenger model than in the baseline model in order to keep the final count of features in challengers' trees identical to the count of features in baselines' trees.

samples. The computational complexity of updating these 3 variables, when a new feature is added, is $\mathscr{O}(d)$ since `treeCount` is a constant.

Limitation Our experiment suffers from one limitation: while we make sure that the feature selection rate is uniform, we ignore interactions between the features. This could be a topic of further research.

Extension One of possible extensions of our work, which we did not pursue further, is pruning of the oldest trees from the ensemble. The idea is simple: the older generations of the trees have so small weight, that they hardly influence the final prediction.

4 Conclusion

We have extended random forest to work on a stream of features. The idea was simple: when a new feature arrives, extend the forest with a new set of trees. However, with this strategy, older features end up used more frequently than the new features. When we fix this feature selection bias, it improves the testing AUC on average by 2 percent points. The proposed algorithm for feature selection bias correction is fast, easy to implement and robust. The code was open-sourced at `https://github.com/janmotl/rf`.

5 Acknowledgments

We would like to thank the anonymous reviewers, their comments helped to improve this paper. This research was supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS17/210/OHK3/3T/18.

References

[1] Leo Breiman. Random forest. *Mach. Learn.*, 45(5):1–35, 1999.

[2] Valentin Kassarnig and Franz Wotawa. Evolutionary propositionalization of multi-relational data. *Proc. 30th Int. Conf. Softw. Eng. Knowl. Eng.*, 2018:629–690, 2018.

[3] Mark-André Krogel. *On Propositionalization for Knowledge Discovery in Relational Databases*. PhD thesis, Otto-von-Guericke-Universität Magdeburg, 2005.

[4] Jan Motl and Pavel Kordík. Do we need to observe features to perform feature selection? *CEUR Workshop Proc.*, 2203:44–51, 2018.

[5] Philipp Probst, Marvin N. Wright, and Anne Laure Boulesteix. Hyperparameters and tuning strategies for random forest, 2019.

[6] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. OpenML: networked science in machine learning. *ACM SIGKDD Explor. Newsl.*, 15(2):49–60, jun 2014.

A Used datasets

2dplanes	biomed	echoMonths	house_8L	rabe_166
abalone	blogger	ecoli	houses	rabe_176
acute-inflammations	blood-transfusion	electricity	humandevel	rabe_265
aids	BNG(breast-w)	elusage	hungarian	rabe_266
Amazon_employee_access	BNG(tic-tac-toe)	fertility	hutsof99_logis	rabe_97
analcatdata_apnea1	bolts	fishcatch	ilpd	rmftsa_ctoarrivals
analcatdata_apnea2	boston	fri_c0_100_10	iris	rmftsa_ladata
analcatdata_apnea3	braziltourism	fri_c0_100_5	irish	rmftsa_sleepdata
analcatdata_asbestos	breast-cancer	fri_c0_1000_10	jEdit_4.0_4.2	Run_or_walk_information
analcatdata_bankruptcy	breast-cancer-dropped	fri_c0_1000_5	jEdit_4.2_4.3	sa-heart
analcatdata_birthday	breast-w	fri_c0_250_10	kdd_el_nino-small	schlvote
analcatdata_bondrate	breastTumor	fri_c0_250_5	kidney	sensory
analcatdata_boxing1	bridges	fri_c0_500_10	kin8nm	servo
analcatdata_boxing2	car	fri_c0_500_5	lowbwt	sleep
analcatdata_broadway	cars	fri_c1_100_10	lupus	sleuth_case1102
analcatdata_broadwaymult	chatfield_4	fri_c1_100_5	machine_cpu	sleuth_case1201
analcatdata_challenger	cholesterol	fri_c1_1000_10	MagicTelescope	sleuth_case1202
analcatdata_chlamydia	chscase_adopt	fri_c1_1000_5	mammography	sleuth_case2002
analcatdata_creditscore	chscase_census2	fri_c1_250_10	mbagrade	sleuth_ex1221
analcatdata_cyyoung8092	chscase_census3	fri_c1_250_5	mfeat-morphological	sleuth_ex1605
analcatdata_cyyoung9302	chscase_census4	fri_c1_500_10	mofn-3-7-10	sleuth_ex1714
analcatdata_dmft	chscase_census5	fri_c1_500_5	monks-problems-1	sleuth_ex2015
analcatdata_draft	chscase_census6	fri_c2_100_10	monks-problems-2	sleuth_ex2016
analcatdata_fraud	chscase_funds	fri_c2_100_5	monks-problems-3	socmob
analcatdata_germangss	chscase_geyser1	fri_c2_1000_10	mozilla4	solar-flare
analcatdata_gsssexsurvey	chscase_health	fri_c2_1000_5	mu284	space_ga
analcatdata_gviolence	chscase_vine1	fri_c2_250_10	mux6	stock
analcatdata_japansolvent	chscase_vine2	fri_c2_250_5	mv	strikes
analcatdata_lawsuit	chscase_whale	fri_c2_500_10	newton_hema	tae
analcatdata_michiganacc	cleve	fri_c2_500_5	no2	threeOf9
analcatdata_neavote	cleveland	fri_c3_100_10	nursery	tic-tac-toe
analcatdata_negotiation	Click_prediction_small	fri_c3_100_5	page-blocks	Titanic
analcatdata_olympic2000	cloud	fri_c3_1000_10	parity5	transplant
analcatdata_reviewer	cm1_req	fri_c3_1000_5	parity5_plus_5	vertebra-column
analcatdata_runshoes	cmc	fri_c3_250_10	pc1_req	veteran
arsenic-female-bladder	datatrieve	fri_c4_250_10	pollen	visualizing_hamster
arsenic-female-lung	delta_ailerons	fri_c4_500_10	postoperative-patient-data	visualizing_livestock
arsenic-male-bladder	delta_elevators	fried	prnn_crabs	visualizing_slope
arsenic-male-lung	diabetes	fruitfly	prnn_fglass	visualizing_soil
autoMpg	diabetes_numeric	glass	prnn_synth	vowel
badges2	diggle_table_a1	grub-damage	profb	wholesale-customers
balance-scale	diggle_table_a2	haberman	puma8NH	wilt
balloon	disclosure_x_bias	hayes-roth	pwLinear	wine
banana	disclosure_x_noise	heart-c	quake	witmer_census_1980
bank8FM	disclosure_x_tampered	heart-h	qualitative-bankruptcy	
banknote-authentication	disclosure_z	heart-statlog	rabe_131	
baskball	dresses-sales	hip	rabe_148	

Table 1: List of used data sets.