Performance and efficacy simulations of the **mlpack** Hoeffding tree

Ryan R. Curtin and Jugal Parikh

November 24, 2015

1 Introduction

The Hoeffding tree (or 'streaming decision tree') is a decision tree induction technique that builds a decision tree on streaming data [2]. The general idea is that Hoeffding bounds can be used to select high-confidence splitting features on the fly, before seeing all samples. This makes the technique applicable to especially large datasets.

However, the available implementation in VFML [4] is not particularly robust or extensible; therefore, I have implemented the Hoeffding tree in **mlpack** [1]. Documentation for how to use the **mlpack** implementation is contained inside the library, and not here.

This document is concerned with benchmarking the performance and efficacy of the **mlpack** Hoeffding tree implementation. Because these performance numbers will change over time, this document is organized in a time-ordered fashion, with the oldest simulations and results first, followed by documentation of the changes performed, followed by more simulations.

2 Week of Nov. 1 2015: batch mode

This week has seen the implementation of 'batch mode' learning, wherein the training set is seen in full before each split of the tree (so the Hoeffding bound is ignored), and then the training set is recursively passed to the leaves, which then split after they have seen the full training set that falls into the region represented by that leaf. This process then repeats until there is no longer any benefit from splitting (according to information gain or Gini impurity, or whatever the fitness function is), or until the leaves see fewer than some number of points (call this l, the 'leaf size').

The goal of this process was to improve the efficacy of Hoeffding trees to come closer to the results of C5.0. Informally, on the `covertype` dataset [5], C5.0 achieves something like 5% test error, whereas streaming-mode Hoeffding trees appeared to achieve more like 20-40% test error, which is less good.

For testing batch mode, which for now we will do only on the 581012x54 `covertype` dataset, we will split the dataset into 75% training set, 25% testing set, and then train the tree, giving the resulting accuracy on the training set and on the test set. We have a handful of parameters we can vary:

- The training mode: 'batch' or 'streaming'
- The numeric feature splitting strategy: 'domingos' (the original), or 'binary' (a slower but better split [3])
- When in batch mode, the leaf size l
- When in streaming mode, the number of passes over the training set p
- When in streaming mode, the confidence c before a split

A note about the leaf size l: a leaf size of l does not mean that every leaf in the decision tree has seen l or fewer training points. Instead, that is merely a limit for splitting. Nodes may not be split at higher levels if it does not give any gain. l simply sets a bound on how much a node can split.

While obtaining data with a leaf size of 1 using the 'binary' split, extreme variance in test set error and overfitting was seen. This is not particularly surprising; Table 1 shows the high variability of test errors.

trial	training accuracy	test accuracy
0	99.097%	69.102%
1	99.097%	60.758%
2	99.097%	79.002%
3	99.097%	63.135%
4	99.097%	58.582%
5	99.097%	70.016%
6	99.081%	84.367%
7	99.081%	62.342%
8	99.081%	92.707%
9	99.081%	93.025%

Table 1: Training and test accuracies for ten trials of the Hoeffding tree with the 'binary' split with $l = 1$; 75%/25% train/test split, on the `covertype` dataset.

Building on this, Table 2 reports simulation results on the `covertype` dataset for different splits, training strategies, l, and p.[1] In these simulations, c was not varied, because the goal here is to determine when overfitting is occuring, and if it can be mitigated.

[1]Due to the binning strategy of the 'domingos' split, it does not make sense to do very small l: the strategy looks at l points, then splits into m equal-sized bins (in these simulations $m = 10$, arbitrarily) based on the smallest and largest of the l points seen so far. So $l = 1$ gives zero-width bins, and very small l is not likely to give decent bins. For streaming mode, the strategy looks at 100 points before binning (again, arbitrary).

split	mode	l	p	training accuracy	testing accuracy
domingos	streaming	–	1	$72.291\% \pm 00.883\%$	$69.217\% \pm 03.281\%$
domingos	streaming	–	2	$74.035\% \pm 00.741\%$	$72.101\% \pm 02.146\%$
domingos	streaming	–	3	$75.458\% \pm 00.563\%$	$72.689\% \pm 01.622\%$
domingos	streaming	–	4	$77.054\% \pm 00.722\%$	$71.921\% \pm 02.370\%$
domingos	streaming	–	5	$78.177\% \pm 00.625\%$	$73.251\% \pm 03.402\%$
domingos	streaming	–	10	$81.237\% \pm 00.478\%$	$77.135\% \pm 02.980\%$
domingos	streaming	–	25	$85.533\% \pm 00.349\%$	$79.331\% \pm 04.436\%$
domingos	streaming	–	50	$88.943\% \pm 00.356\%$	$81.096\% \pm 04.558\%$
domingos	streaming	–	100	$91.872\% \pm 00.282\%$	$77.199\% \pm 03.695\%$
binary	streaming	–	1	$72.138\% \pm 00.291\%$	$66.506\% \pm 04.859\%$
binary	streaming	–	2	$74.296\% \pm 00.178\%$	$62.952\% \pm 07.335\%$
binary	streaming	–	3	$75.644\% \pm 00.236\%$	$65.761\% \pm 05.099\%$
binary	streaming	–	4	$76.706\% \pm 00.425\%$	$65.226\% \pm 04.803\%$
binary	streaming	–	5	$77.540\% \pm 00.466\%$	$63.710\% \pm 05.756\%$
binary	streaming	–	10	$80.891\% \pm 00.263\%$	$58.031\% \pm 08.406\%$
binary	streaming	–	25	$85.967\% \pm 00.157\%$	$65.151\% \pm 09.045\%$
binary	streaming	–	50	$89.825\% \pm 00.200\%$	$72.157\% \pm 11.889\%$
binary	streaming	–	100	$93.201\% \pm 00.105\%$	$76.666\% \pm 11.431\%$
domingos	batch	5	–	$97.330\% \pm 00.156\%$	$75.406\% \pm 06.234\%$
domingos	batch	10	–	$95.393\% \pm 00.240\%$	$79.543\% \pm 05.050\%$
domingos	batch	25	–	$91.493\% \pm 00.328\%$	$78.392\% \pm 03.899\%$
domingos	batch	50	–	$88.370\% \pm 00.364\%$	$79.322\% \pm 04.298\%$
domingos	batch	100	–	$84.870\% \pm 00.381\%$	$78.114\% \pm 04.273\%$
domingos	batch	250	–	$80.233\% \pm 00.337\%$	$75.909\% \pm 03.109\%$
domingos	batch	500	–	$76.948\% \pm 00.516\%$	$71.594\% \pm 02.915\%$
domingos	batch	1000	–	$74.928\% \pm 00.759\%$	$73.510\% \pm 09.285\%$
binary	batch	1	–	$99.081\% \pm 00.001\%$	$77.130\% \pm 12.152\%$
binary	batch	2	–	$99.091\% \pm 00.000\%$	$79.796\% \pm 09.769\%$
binary	batch	5	–	$99.115\% \pm 00.000\%$	$73.261\% \pm 15.864\%$
binary	batch	10	–	$98.094\% \pm 00.000\%$	$68.779\% \pm 07.513\%$
binary	batch	25	–	$95.943\% \pm 00.000\%$	$69.865\% \pm 10.309\%$
binary	batch	50	–	$93.569\% \pm 00.126\%$	$70.853\% \pm 13.196\%$
binary	batch	100	–	$90.872\% \pm 00.000\%$	$70.455\% \pm 11.724\%$
binary	batch	250	–	$86.336\% \pm 00.000\%$	$67.873\% \pm 07.003\%$
binary	batch	500	–	$83.181\% \pm 00.000\%$	$69.625\% \pm 07.065\%$
binary	batch	1000	–	$79.809\% \pm 00.000\%$	$64.902\% \pm 03.419\%$

Table 2: Hoeffding tree efficacy on the `covertype` dataset; 75%/25% train/test split; 10 trials.

These results are not what I expected. They make some amount of sense for the 'domingos' split, but for the 'binary' split, it really appears like something is wrong, like a bug in the program; the test accuracy is never anywhere near close to the training accuracy—especially in the streaming setting. The standard deviation is also huge, making the 'binary' results hard to interpret. A worthwhile next thing to do, then, will be to compare these results with the VFML implementation.

3 Week of Nov 8 2015: VFML and C5.0 comparison

In order to make sense of the results above, it would be prudent to compare against the VFML implementation to ensure that the results are similar.

mode	p	training accuracy	testing accuracy
streaming	1	$68.215\% \pm 0.495\%$	$68.228\% \pm 0.599\%$
streaming	2	$69.340\% \pm 0.253\%$	$69.346\% \pm 0.274\%$
streaming	3	$70.699\% \pm 0.309\%$	$70.606\% \pm 0.293\%$
streaming	4	$71.553\% \pm 0.293\%$	$71.478\% \pm 0.283\%$
streaming	5	$72.091\% \pm 0.177\%$	$72.031\% \pm 0.212\%$
streaming	10	$73.819\% \pm 0.195\%$	$73.662\% \pm 0.132\%$
streaming	25	$76.919\% \pm 0.230\%$	$76.599\% \pm 0.256\%$
streaming	50	$79.424\% \pm 0.220\%$	$78.914\% \pm 0.277\%$
streaming	100	$82.873\% \pm 0.213\%$	$82.077\% \pm 0.222\%$
streaming	250	$87.844\% \pm 0.118\%$	$86.385\% \pm 0.169\%$
streaming	500	$91.563\% \pm 0.106\%$	$89.286\% \pm 0.104\%$
batch	–	$59.753\% \pm 9.750\%$	

Table 3: Results of VFML toolkit in streaming and batch mode, with 75%/25% train/test split. l is pegged to 5 for batch mode.

In addition, to set the goals for what kind of accuracy we are looking for, we can compare against the GPL C5.0 implementation. In Table 3, results are given for C5.0. C5.0 also supports bagging, so the parameter t, the number of trees, is swept across several values. Pre-pruning is also an option; this specifies the minimum number p of training points a node must see to be included in the tree; this presumably prevents overfitting. In these trials, only one run was performed, so variance numbers do not exist.

These results from C5.0, and the earlier results from VFML, are a little more in line with the expected results, with significantly less noise. This means that there is likely to be a bug in the mlpack implementation, and therefore reducing the test variance or replicating the VFML results more precisely is the first priority.

b	p	training accuracy	test accuracy
1	–	98.1%	94.2%
3	–	99.3%	94.8%
4	–	99.5%	95.5%
5	–	99.8%	95.8%
10	–	100.0%	96.7%
25	–	100.0%	97.0%
50	–	100.0%	97.1%
100	–	100.0%	97.2%
1	5	96.9%	93.5%
1	10	95.3%	92.3%
1	25	92.4%	90.3%
1	50	89.5%	88.0%
1	100	86.1%	85.2%
1	250	82.0%	81.5%
1	500	78.2%	77.8%
1	1000	75.6%	75.5%

Table 4: Performance of C5.0 on the covertype dataset, with a 75%/25% train/test split, varying the number of trees in the forest (b) and the pre-pruning parameter p.

4 Week of Nov 15 2015: reducing test variance

A large amount of investigation and searching for bugs has revealed several possibilities for explaining the huge test variance of the **mlpack** implementation and other differences when compared to the VFML implementation:

1. `--max_samples` being used incorrectly and accidentally set artifically too low during the previous tests
2. the use of the Gini impurity as opposed to the information gain
3. allowing only one possible split per dimension, which could cause the Hoeffding bound to choose a split that wasn't the best in a certain dimension
4. a suboptimal numeric split procedure that differs from VFML

Of these, item (1) is easy to fix, item (2) was fixed in `66592fa`, item (3) was fixed in `81a8b6c` and `278a5f8`, and (4) has not yet been approached. However, it turns out that the real issue was that the test data was being loaded improperly. This was fixed in `967e230`. Now, we can actually start generating meaningful results.

References

[1] R.R. Curtin, J.R. Cline, N.P. Slagle, W.B. March, P. Ram, N.A. Mehta, and A.G. Gray. MLPACK: A scalable C++ machine learning library. *The Journal of Machine Learning Research*, 14(1):801–805, 2013.

[2] P. Domingos and G. Hulten. Mining high-speed data streams. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '00)*, pages 71–80. ACM, 2000.

[3] J. Gama, R. Rocha, and P. Medas. Accurate decision trees for mining high-speed data streams. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '03)*, pages 523–528. ACM, 2003.

[4] Geoff Hulten and Pedro Domingos. VFML – a toolkit for mining high-speed time-changing data streams. 2003.

[5] M. Lichman. UCI machine learning repository, 2013.

split	mode	gain	l	p	training accuracy	testing accuracy
domingos	streaming	gini	–	1	$71.220\% \pm 0.486\%$	$71.147\% \pm 0.547\%$
domingos	streaming	gini	–	2	$73.313\% \pm 0.397\%$	$73.072\% \pm 0.486\%$
domingos	streaming	gini	–	3	$74.585\% \pm 0.736\%$	$74.224\% \pm 0.769\%$
domingos	streaming	gini	–	4	$75.291\% \pm 0.361\%$	$74.839\% \pm 0.326\%$
domingos	streaming	gini	–	5	$76.220\% \pm 0.319\%$	$75.684\% \pm 0.402\%$
domingos	streaming	gini	–	10	$78.954\% \pm 0.593\%$	$78.018\% \pm 0.703\%$
domingos	streaming	gini	–	25	$83.759\% \pm 0.635\%$	$81.641\% \pm 0.673\%$
domingos	streaming	gini	–	50	$87.333\% \pm 0.470\%$	$83.675\% \pm 0.681\%$
domingos	streaming	gini	–	100	$91.233\% \pm 0.220\%$	$84.847\% \pm 0.390\%$
domingos	streaming	gini	–	250	$95.018\% \pm 0.199\%$	$84.432\% \pm 0.520\%$
domingos	streaming	gini	–	500	$97.190\% \pm 0.151\%$	$83.068\% \pm 0.834\%$
domingos	streaming	gini	–	1000	$99.157\% \pm 0.029\%$	$82.548\% \pm 0.521\%$
domingos	streaming	info	–	1	$67.897\% \pm 0.278\%$	$67.905\% \pm 0.272\%$
domingos	streaming	info	–	2	$68.421\% \pm 0.320\%$	$68.478\% \pm 0.325\%$
domingos	streaming	info	–	3	$68.876\% \pm 0.260\%$	$68.846\% \pm 0.324\%$
domingos	streaming	info	–	4	$69.422\% \pm 0.574\%$	$69.439\% \pm 0.627\%$
domingos	streaming	info	–	5	$70.066\% \pm 0.263\%$	$70.013\% \pm 0.295\%$
domingos	streaming	info	–	10	$71.508\% \pm 0.599\%$	$71.368\% \pm 0.626\%$
domingos	streaming	info	–	25	$74.729\% \pm 0.572\%$	$74.347\% \pm 0.649\%$
domingos	streaming	info	–	50	$77.710\% \pm 0.421\%$	$77.029\% \pm 0.440\%$
domingos	streaming	info	–	100	$80.886\% \pm 0.641\%$	$79.566\% \pm 0.630\%$
domingos	streaming	info	–	250	$85.656\% \pm 0.487\%$	$82.529\% \pm 0.529\%$
domingos	streaming	info	–	500	$89.247\% \pm 0.325\%$	$84.072\% \pm 0.497\%$
domingos	streaming	info	–	1000	$91.978\% \pm 0.198\%$	$84.545\% \pm 0.383\%$
domingos	batch	gini	5	–	$97.328\% \pm 0.113\%$	$83.066\% \pm 0.621\%$
domingos	batch	gini	10	–	$95.367\% \pm 0.196\%$	$85.846\% \pm 0.579\%$
domingos	batch	gini	25	–	$91.390\% \pm 0.269\%$	$86.529\% \pm 0.404\%$
domingos	batch	gini	50	–	$88.033\% \pm 0.314\%$	$85.127\% \pm 0.322\%$
domingos	batch	gini	100	–	$84.585\% \pm 0.536\%$	$82.888\% \pm 0.612\%$
domingos	batch	gini	250	–	$80.656\% \pm 0.485\%$	$79.785\% \pm 0.531\%$
domingos	batch	gini	500	–	$77.344\% \pm 0.694\%$	$76.934\% \pm 0.755\%$
domingos	batch	gini	1000	–	$74.795\% \pm 0.417\%$	$74.518\% \pm 0.458\%$
domingos	batch	info	5	–	$97.300\% \pm 0.108\%$	$83.159\% \pm 0.542\%$
domingos	batch	info	10	–	$95.347\% \pm 0.158\%$	$86.123\% \pm 0.336\%$
domingos	batch	info	25	–	$91.453\% \pm 0.263\%$	$86.812\% \pm 0.427\%$
domingos	batch	info	50	–	$87.610\% \pm 0.449\%$	$84.825\% \pm 0.597\%$
domingos	batch	info	100	–	$84.358\% \pm 0.664\%$	$82.715\% \pm 0.698\%$
domingos	batch	info	250	–	$79.859\% \pm 0.521\%$	$78.993\% \pm 0.529\%$
domingos	batch	info	500	–	$76.416\% \pm 0.519\%$	$76.008\% \pm 0.549\%$
domingos	batch	info	1000	–	$74.373\% \pm 0.420\%$	$74.152\% \pm 0.401\%$

Table 5: Hoeffding tree efficacy on the `covertype` dataset; 75%/25% train/test split; 10 trials. Git revision `0e93455`.

split	mode	gain	l	p	training accuracy	testing accuracy
binary	streaming	gini	–	1	$73.362\% \pm 0.243\%$	$73.264\% \pm 0.264\%$
binary	streaming	gini	–	2	$75.506\% \pm 0.134\%$	$75.220\% \pm 0.154\%$
binary	streaming	gini	–	3	$76.961\% \pm 0.175\%$	$76.763\% \pm 0.212\%$
binary	streaming	gini	–	4	$78.046\% \pm 0.262\%$	$77.697\% \pm 0.269\%$
binary	streaming	gini	–	5	$79.032\% \pm 0.171\%$	$78.652\% \pm 0.176\%$
binary	streaming	gini	–	10	$82.463\% \pm 0.238\%$	$81.710\% \pm 0.234\%$
binary	streaming	gini	–	25	$86.994\% \pm 0.149\%$	$85.579\% \pm 0.185\%$
binary	streaming	gini	–	50	$90.565\% \pm 0.129\%$	$88.344\% \pm 0.177\%$
binary	streaming	gini	–	100	$94.001\% \pm 0.126\%$	$90.721\% \pm 0.184\%$
binary	streaming	gini	–	250	fail	fail
binary	streaming	gini	–	500	fail	fail
binary	streaming	gini	–	1000	fail	fail
binary	streaming	info	–	1	$67.823\% \pm 0.328\%$	$67.712\% \pm 0.363\%$
binary	streaming	info	–	2	$68.899\% \pm 0.106\%$	$68.831\% \pm 0.170\%$
binary	streaming	info	–	3	$69.839\% \pm 0.455\%$	$69.769\% \pm 0.439\%$
binary	streaming	info	–	4	$70.795\% \pm 0.203\%$	$70.713\% \pm 0.226\%$
binary	streaming	info	–	5	$71.412\% \pm 0.121\%$	$71.322\% \pm 0.200\%$
binary	streaming	info	–	10	$72.990\% \pm 0.190\%$	$72.921\% \pm 0.201\%$
binary	streaming	info	–	25	$75.443\% \pm 0.122\%$	$75.271\% \pm 0.179\%$
binary	streaming	info	–	50	$78.227\% \pm 0.147\%$	$77.851\% \pm 0.152\%$
binary	streaming	info	–	100	$81.491\% \pm 0.180\%$	$80.812\% \pm 0.178\%$
binary	streaming	info	–	250	fail	fail
binary	streaming	info	–	500	fail	fail
binary	streaming	info	–	1000	fail	fail
binary	batch	gini	1	–	$99.098\% \pm 0.016\%$	$93.061\% \pm 0.085\%$
binary	batch	gini	2	–	$99.105\% \pm 0.009\%$	$93.092\% \pm 0.086\%$
binary	batch	gini	5	–	$99.090\% \pm 0.017\%$	$93.019\% \pm 0.110\%$
binary	batch	gini	10	–	$98.085\% \pm 0.019\%$	$92.765\% \pm 0.062\%$
binary	batch	gini	25	–	$95.927\% \pm 0.043\%$	$91.833\% \pm 0.097\%$
binary	batch	gini	50	–	$93.604\% \pm 0.072\%$	$90.453\% \pm 0.158\%$
binary	batch	gini	100	–	$90.731\% \pm 0.107\%$	$88.405\% \pm 0.117\%$
binary	batch	gini	250	–	$86.319\% \pm 0.99\%$	$84.900\% \pm 0.143\%$
binary	batch	gini	500	–	$83.055\% \pm 0.087\%$	$82.186\% \pm 0.169\%$
binary	batch	gini	1000	–	$80.104\% \pm 0.149\%$	$79.514\% \pm 0.153\%$
binary	batch	info	1	–	$99.297\% \pm 0.009\%$	$93.661\% \pm 0.086\%$
binary	batch	info	2	–	$99.296\% \pm 0.011\%$	$93.649\% \pm 0.122\%$
binary	batch	info	5	–	$99.301\% \pm 0.019\%$	$93.637\% \pm 0.063\%$
binary	batch	info	10	–	$98.369\% \pm 0.021\%$	$93.381\% \pm 0.039\%$
binary	batch	info	25	–	$96.201\% \pm 0.065\%$	$92.321\% \pm 0.112\%$
binary	batch	info	50	–	$93.814\% \pm 0.037\%$	$90.908\% \pm 0.084\%$
binary	batch	info	100	–	$90.804\% \pm 0.079\%$	$88.645\% \pm 0.126\%$
binary	batch	info	250	–	$86.373\% \pm 0.153\%$	$85.110\% \pm 0.167\%$
binary	batch	info	500	–	$82.798\% \pm 0.186\%$	$81.983\% \pm 0.235\%$
binary	batch	info	1000	–	$79.493\% \pm 0.219\%$	$78.907\% \pm 0.230\%$

Table 6: Hoeffding tree efficacy on the `covertype` dataset; 75%/25% train/test split; 10 trials. Git revision `0e93455`.