Chapter 6

Evaluation

This chapter will give an overview over the achieved results. After covering the methodology and experimental setup, the DP-GBDT performance results are presented. Thereafter, the runtime of the different implementations is compared. This gives an image on the impact of individual hardening measures, and the impact of running the code inside an enclave. The sections are both concluded with a brief discussion.

6.1 Methodology

In this section we demonstrate the performance of the algorithm on four different real world datasets. The datasets vary significantly in size. Two of them are regression tasks, the other ones are binary classification tasks.

name	size	features	task	only use subset
abalone [17]	4177	8	regression	no
yearMSD [18]	515345	90	regression	yes
BCW [20]	699	10	binary classification	no
adult [19]	48842	14	binary classification	no

Table 6.1: Datasets

Experimental Setup The machine that generated these results is running x86_64 GNU/Linux, Debian 10, kernel 4.19.171-2 on an Intel Core i7-7600U @ 2.8 GHz Kaby Lake CPU. It has 20GB RAM, 32KB of L1, 256KB of L2 and 4MB of L3 cache. The compiler is g++/gcc version 8.3.0 (Debian 8.3.0-6).

6.2 Performance

The measurements are carried out for 21 distinct privacy budgets between 0.1 and 10. The number of trees per ensemble and the amount of samples used varies amongst the different datasets. The result values represent the average of running 5-fold cross validation. Therefore a `samples` parameter of 5000 indicates that 4000 samples were used for training and 1000 for validation. The errorbars denote the average standard deviation measured. The regression baseline is achieved by always predicting the average/mean feature value. For binary classification the baseline is

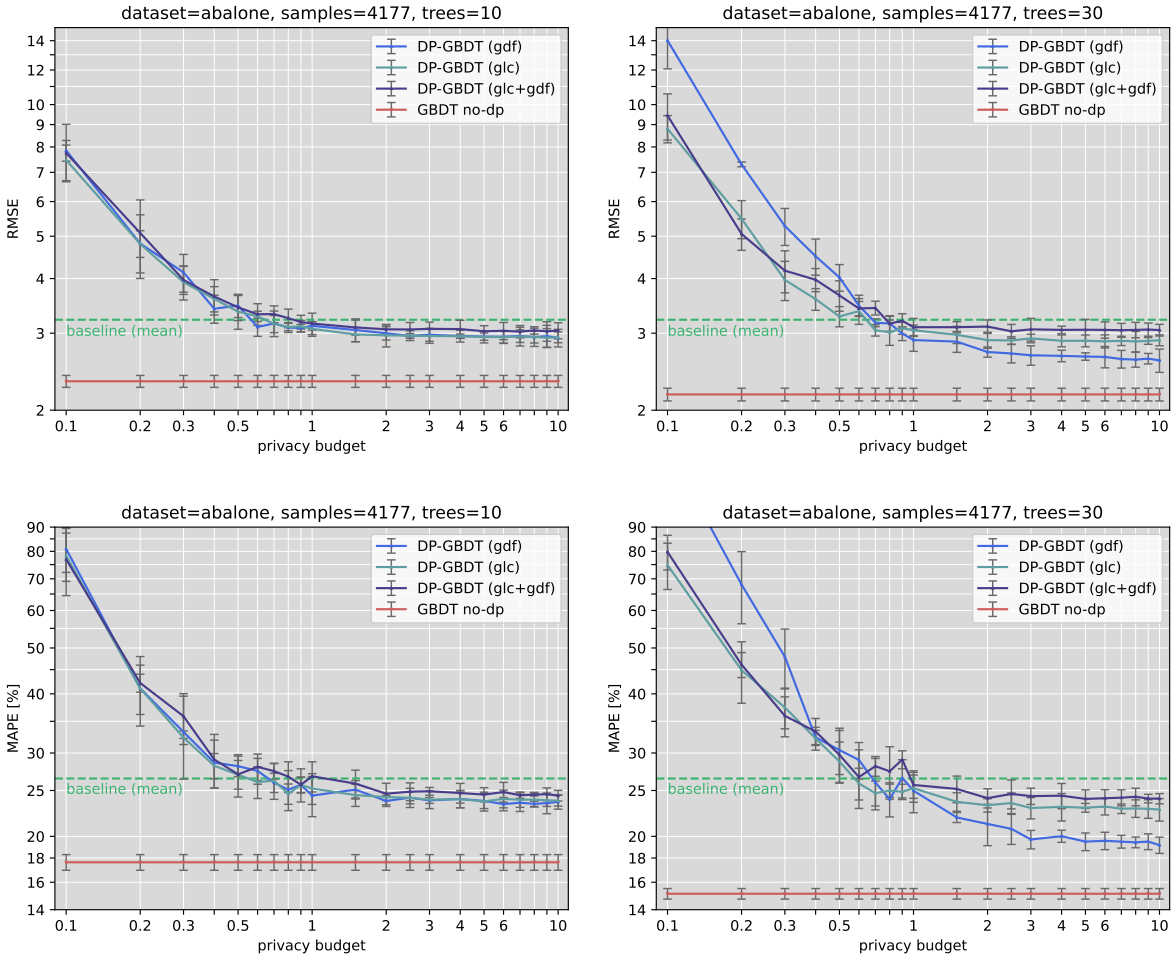

Figure 6.1: Abalone RMSE (top) / MAPE (bottom): Left side ensemble of 10 trees, right side ensemble of 30 trees

attained through the 0R-classifier. For regression tasks we provide both RMSE (penalises errors on outliers) and MAPE (more robust to outliers).

Hyperparameters The utilized hyperparameters, are listed in appendix A. The `use_grid` option is turned off, since it leads to virtually the same results given a small enough step size while substantially increasing runtime. Experiments showed that our way of scaling feature values into the grid (details can be found in paragraph 4.3) works reliably using a privacy budget of just 0.05. However, there are certainly more effective ways of performing this task that cost less privacy budget (e.g. [16]). Further, note that the chosen hyperparameters are **by no means optimal** in terms of performance. With hyperparameter tuning, even better results should easily be achievable. Such tuning was omitted due to timing constraints and due to the lack of suitable frameworks for this task in C++.

Abalone For abalone we can take the full dataset (4177 samples). As it is a regression task we can take the mean of the target feature as a baseline, which in the case of abalone is around 3.22. Figure 6.1 shows that all DP-GBDT variations start defeating the baseline at a privacy budget of around 0.7. Using more trees (30 instead of 10) clearly improves performance. The right

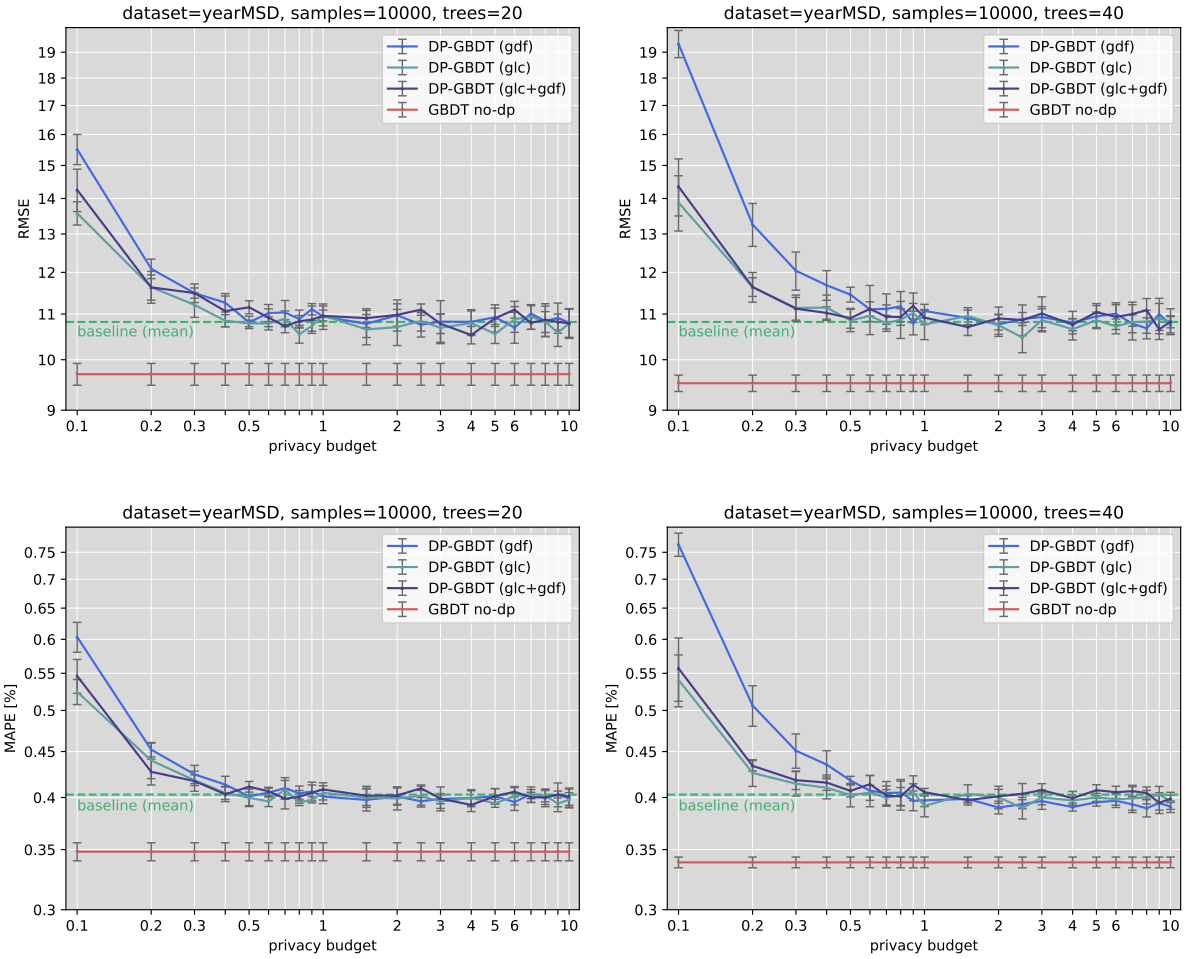

Figure 6.2: Year RMSE (top) / MAPE (bottom): Left side ensemble of 20 trees, right side ensemble of 40 trees

hand side plots also show that using GDF leads to worse for small privacy budgets, but better performance for higher budgets. At no point however, do the DP-GBDT implementations come reasonable close to the non-dp performance. GLC does not seem to improve performance in any form. With the RMSE and MAPE looking very similar, it seems like outliers are forecasted pretty effectively.

yearMSD Since this dataset is quite large (over 500k entries and around 90 features) experiments were conducted on subsets of it. Further, following the guidelines on its source webpage, it was ensured that training and test samples were chosen from distinct parts of the dataset. This avoids the 'producer effect', so no song from a given artist ends up in both the train and test set. Figure 6.2 shows that using a subset of 10k samples is just not enough to reliably beat the baseline. Creating ensembles with 40 instead of 20 trees shows no significant impact. As with the previous dataset, usage of GDF tends to negatively impact performance for small privacy budgets.

Breast Cancer Wisconsin Being the smallest dataset, the results were not as stable as those from other datasets. Therefore the result values are the average of 5-fold cross validation repeated 10 times. Figure 6.3 shows that all DP-GBDT variations start defeating the baseline at a privacy

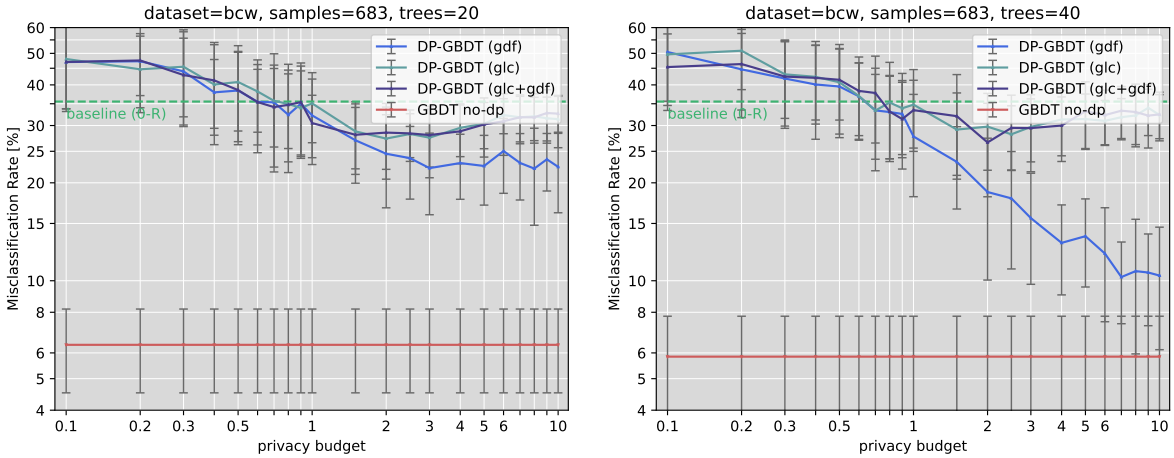

Figure 6.3: BCW misclassification rate: left side ensemble of 20 trees, right side 40 trees

budget of around 0.7. The GDF variation is the only version that is able to continuously improve its score with increasing privacy budget. The other variants achieve their best scores at privacy budgets of around 3, afterwards, their results get worse. Using more trees (40 instead of 20) improves the performance mainly when using GDF. Again, GLC does not seem to improve performance in any significant manner.

Adult Using this dataset the task is to predict whether an individual's annual income exceeds $50,000 based on census data. Other machine learning algorithms' scores are generally in the range of 14-20% error rate [38]. Figure 6.4 shows that the DP-GBDT algorithm does not really

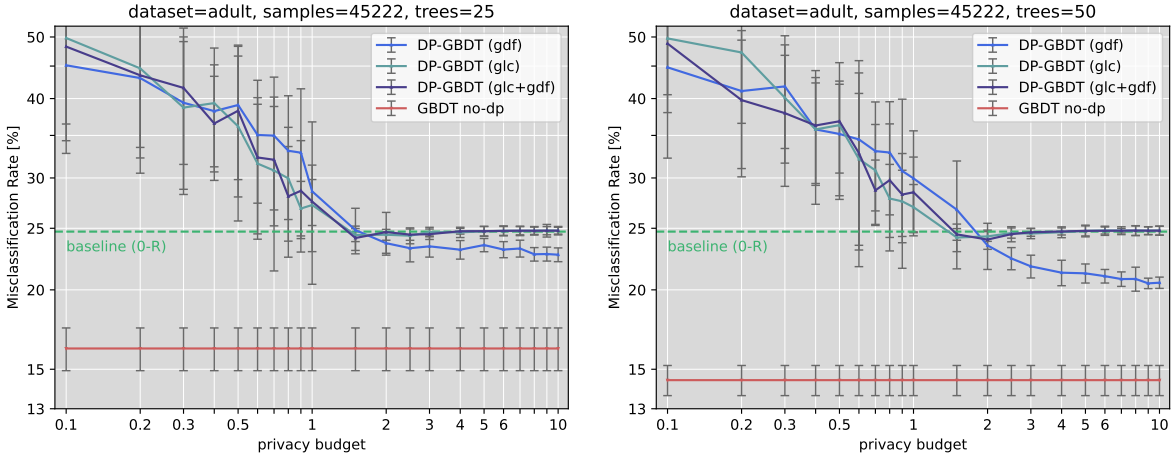

Figure 6.4: Adult misclassification rate: left side ensemble of 25 trees, right side 50 trees

work well on this dataset. At least not with these specific hyperparameters. Only the GDF version manages to beat the baseline. Not by a big margin and only for ε's ≥ 2. Once again, increasing the amount of trees in the ensemble from 25 to 50 does not have a great impact.

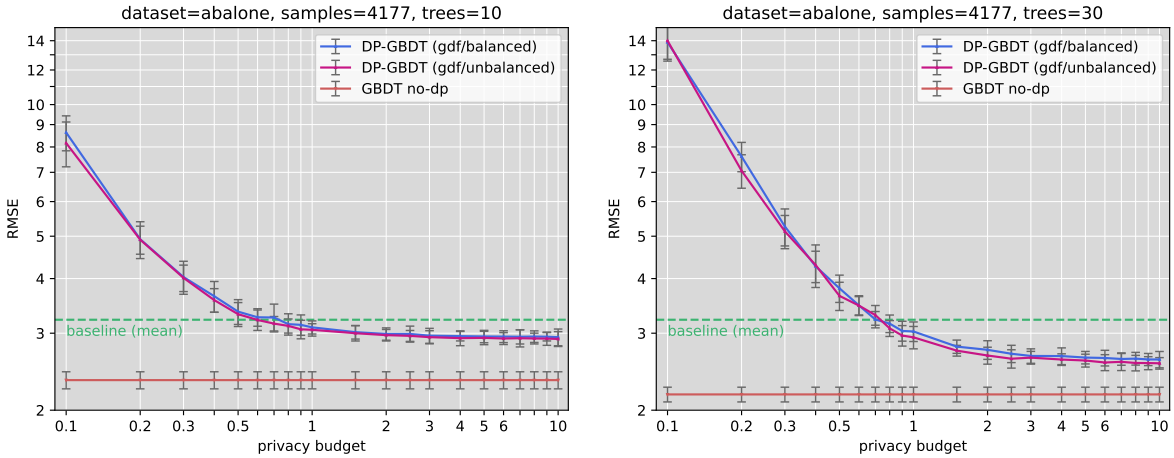

Figure 6.5: Abalone RMSE: Balanced/unbalanced comparison: ensemble of 10 resp. 30 trees

Balanced sample distribution This paragraph shows the effect of a balanced vs. unbalanced sample distribution strategy using the abalone dataset as an example. The unbalanced variant uses the formula in (TODO ref to background) to distribute samples (the number is always decreasing). The balanced version hands out the same number of samples to all trees in an ensemble. As depicted in figure 6.5, using a balanced instead of an unbalanced approach does not significantly affect performance.

6.2.1 Discussion

First of all, it's interesting that, unlike DPBoost [40] proclaims, GLC did not improve the overall performance on any of our datasets. The performance was notably worse for the adult dataset. It's likely that this is at least partially due to a general weakness of many decision tree algorithms: Decision tree learners can create biased trees if some classes dominate [42], which is the case for the adult dataset (25%/75%). Balancing the dataset prior to fitting might improve this result. In general, performance for small privacy budgets is quite bad. We seem to be adding so much noise, that the use of trees of our depth (generally a depth of 6 was used) does more harm than good. So it seems like small $\varepsilon's$ are actually a real problem for decision trees. As the overlying goal is to achieve a useful learning process for privacy budgets around 0.1 or 0.2, there's still quite some work to do. But a couple of improvements down the line this might be possible. Also keep in mind, that the current design's performance can certainly be vastly improved by well-versed hyperparameter tuning.

6.3 Runtime

In this section we demonstrate the training time variations between the different implementations. Even the DP-GBDT algorithm will likely not be run very frequently in practice, achieving a respectable runtime performance was important to us.

Hyperparameters We use the following parameters for all measurements across all implementations:

privacy_budget	0.5	gradient_filtering	false
nb_trees	10	leaf_clipping	true
min_samples_split	2	balance_partition	true
learning_rate	0.1	use_grid	false
max_depth	6	use_dp	true

Table 6.2: Parameters for runtime measurements

Results All runtime results were obtained by performing 5-fold cross validation. Therefore the the measured values in 6.3 have been divided by 5.

python_gbdt: Patched version of the reference DP-GBDT implementation that was created by T. Giovanna prior to this thesis. It utilizes multithreading through `sklearn`.

cpp_gbdt: Base C++ version that was extensively described in section 4.2. It does not run inside an enclave and it is not hardened. It mainly exists for experimentation with the underlying algorithm. Several measures (e.g. multithreading) have been undertaken to boost its performance (details can be found in 4.2).

enclave_gbdt: This is essentially `cpp_gbdt` but ported into an SGX enclave.

hardened_gbdt: Hardened version of `cpp_gbdt`. It is not placed in an enclave however. Chapter 5 described the hardening process. Even though no side-channel violations were found when compiling with the `-O1` flag, we included the `-O0` measurement as well to be on the safe side.

hardened_sgx_gbdt: This is the final combination of `hardened_gbdt` and `enclave_gbdt`.

Implementation	**abalone**			**adult**			**yearMSD**	
	300	**1000**	**4177**	**300**	**1000**	**5000**	**300**	**1000**
python_gbdt	1.5	2.3	15.2	1.5	3.4	41.8	9.4	88.3
cpp_gbdt	<0.01	0.01	0.05	<0.01	0.01	<0.01	<0.01	0.1
enclave_gbdt	0.05	0.26	1.38	todo	todo	todo	todo	todo
hardened_gbdt -O0	1.04	10.3	154.6	1.94	15.8	366	11.9	102
hardened_gbdt -O1	0.18	1.6	22.0	0.32	2.46	59.5	2.0	16.2
hardened_sgx_gbdt	todo	todo	todo	todo	todo	todo	todo	todo

Table 6.3: Runtime results [s]

First of all, compared to the Python reference implementation, `cpp_gbdt` offers a solid 506x speedup on average. The speedup increases further with even larger datasets. This is because its bottleneck in training is finding good splits (see B.1), and this task grows non-linearly with increasing dataset size. Moving `cpp_gbdt` into an enclave, decreases its runtime by [TODO]x on average. This can certainly be contributed to the absence of multithreading, but also to general SGX related overhead like encryption and enclave calls. `hardened_gbdt -O1` is on average 381x slower than `cpp_gbdt`. Turning off compiler optimizations (compile with `-O0`) further increases the runtime by 6.3x on average, which results in a total slowdown of around 2390x. But note that this implementation contains a few more hardening measures than strictly necessary for differential privacy (see 5.2.2). Last but not least, `hardened_sgx_gbdt`, the combination of hardened DP-GBDT and running in an SGX enclave, is as expected once again noticeably slower, [TODO]x on average.

Next, table 6.4 shows the runtime consequences of grid-usage and use of constant-time fixed point arithmetic. For the grid, a step size of [TODO] was employed. To demonstrate the effect of using constant-time fixed point numbers/operations instead of floating point arithmetic, we replaced them in the core functions of the algorithm. Around 95% of execution time is spent these functions (see B).

Implementation	**abalone**			**adult**			**yearMSD**	
	300	**1000**	**4177**	**300**	**1000**	**5000**	**300**	**1000**
h_1 := hardened_gbdt -O1	0.18	1.6	22.0	0.32	2.46	59.5	2.0	16.2
h_1 + use_grid	todo	todo	todo	todo	todo	todo	todo	todo
h_1 + libfixedtimefixedpoint	todo	todo	todo	todo	todo	todo	todo	todo

Table 6.4: Runtime implications of `use_grid` and using constant-time fixed point arithemtic [s]

The results show that. TODO "grid is a big overhead, lftfp is ok."

6.3.1 Discussion

As our runtime results show, hardening can impose a lot of runtime overhead. Especially when a "complete" hardening approach is chosen. If a more minimal way of hardening is applied, like in the our DP-GBDT case, it's much more efficient. In hindsight, at the time of writing this thesis, also a few weaknesses of `hardened_gbdt` became more and more apparent. For example, using BFS instead of DFS tree induction could allow significant execution time improvements for the hardened version. This shows the importance of already reflecting about constant-time properties very early on in such a project. We further showed that porting the DP-GBDT algorithm into the enclave introduced a very manageable overhead. We also produced a fast baseline implementation, ideal for future experiments. In general our runtime experiments underline the feasibility of this setup for future usage in practice.