A comparison of CUDA-accelerated SVM, random forest, and logistic regression binary classification performance

Jitarth Sheth
COGS 118 - Supervised Machine Learning Algorithms
University of California - San Diego
La Jolla, CA 92093
jhsheth@ucsd.edu

Abstract

Caruana and Niculescu-Mizil [2006] conducted an empirical comparison of supervised machine learning algorithms as a follow up to a previous study, STATLOG [King et al., 2000]. In the time since this comparison, implementations of these algorithms have greatly changed. Additionally, the development of CUDA-acceleration has made quick training of large data sets feasible. This paper attempts to replicate the results of Caruana and Niculescu-Mizil [2006] by running a subset of algorithms (SVMs, random forests, and logistic regression) on new, modern data sets. The scores of each algorithm over multiple metrics and data sets are presented. This paper also explores the broader implications of GPU machine learning and attempts to quantify training speed over traditional CPU machine learning.

1 Introduction

Acknowledging the few studies conducted comparing various supervised learning algorithms, Caruana and Niculescu-Mizil [2006], referred to as CNM06, conducted an empirical comparison of supervised learning algorithms. CNM06 followed a similar comparison, STATLOG [King et al., 2000], and reevaluated algorithms using models which were modern at the time.

CNM06 [2006] explores computationally feasible parameters for each algorithm by initiating a hyperparameter search and choosing the hyperparameters with the best performance on training sets with each chosen metric. Training sets consist of 5000 samples from well-known data sets. As metrics for evaluation, CNM06 measures accuracy, F-score, lift, and area under the ROC curve, among others. CNM06 concludes that the algorithm with the best performance is calibrated boosted trees, followed by random forests, uncalibrated bagged trees, calibrated SVMs, and uncalibrated neural nets. However, the results of CNM06 may not prevail with modern development of the algorithms presented within the paper. The most notable being the improvement of algorithm training and performance through graphics processing unit (GPU) acceleration.

As a follow up, this paper evaluates a subset of the algorithms of CNM06 [2006]: SVMs, random forests, and logistic regression. SVMs and random forests were chosen as they were among the highest performing algorithms in CNM06. In contrast to CNM06, these algorithms are not calibrated. These algorithms are also evaluated on a subset of metrics and a smaller scale of data sets are also chosen. Overall, this paper completes a modified replication of CNM06, with certain limitations.

2 Methodology

We conduct a hyperparameter grid search for each algorithm to choose the best hyperparameters for each metric. Hyperparameters are tuned using 5-fold cross validation on training sets.

2.1 Learning Algorithms

SVMs: we use a radial basis function kernel in cuML's [Raschka et al., 2020] implementation of the sequential minimal optimization (SMO) problem, similar to ThunderSVM's [Wen et al., 2018] implementation. We vary the inverse regularization penalty parameter C from 10^{-7} to 10^{3} and the kernel coefficient parameter $gamma$ from 10^{-6} to 10^{-2} with an additional $gamma$ parameter of $\frac{1}{\#features}$. Linear and polynomial kernels are not explored in this analysis due to the large cost of increased complexity and running time on each data set.

Random Forests (RF): we use cuML's implementation of a random forest classifier with total tree estimators of {128, 256, 512, 1024} and ratio of number of features to consider per node split of $\{\frac{1}{\sqrt{\#features}}, \frac{1}{\log_2(\#features)}\}$. We use a Gini impurity function to measure the quality of splits.

Logistic Regression (LOG-REG): we use cuML's implementation of a logistic regression classifier with solvers of L-BGFS or OWL-QN. We train unregularized, L1 regularized, and L2 regularized models. We also vary the inverse regularization penalty parameter C from 10^{-8} to 10^{4}.

2.2 Performance Metrics

We evaluate algorithm performance with the threshold metrics of accuracy (ACC), F-score (FSC). We also evaluate algorithms with the rank metric of area under the ROC curve (AUC). All metrics range from [0,1], with higher values indicating higher performance for ACC and FSC. These metrics are used for evaluation of training sets, tuning hyperparameters through 5-fold cross validation, and evaluation of test sets.

2.3 Data Sets

Algorithms are evaluated on performance on four different data sets: EEG, OCC, LEA, and CSG. The EEG and OCC data sets were obtained from the UCI Machine Learning Repository [Dua and Graff, 2017]. The EEG data set consists of measurements from a neuroheadset with a binary classification label of eye-open or eye-closed. The OCC data set consists of binary labeled occupancy detection data, along with various room sensor values. The feature of timestamp was disregarded in this data set. The LEA data set consists of data from various games of *League of Legends* obtained on Kaggle [Riot, 2020] with binary labeled data of blue team win or loss. The CSG data set consists of round data from game matches of *Counter Strike: Global Offensive* obtained on Kaggle [Valve and Machado, 2020] with binary labeled data of round winner. We set round winner T to 0 and CT to 1. The feature of map_name was one-hot encoded for accurate training. The LEA and CSG data sets were selected as they represent modern data from non-laboratory sources and contain a large number of trainable features. Each data set was cleaned and scaled according to the data set's interquartile range using a robust scaler to prevent the influence of large outliers. EEG, LEA, and CSG are well balanced problems, with positive labels consisting of around 50% of each data set.

Table 1: Description of data sets

DATASET	#ATTR	TRAIN SIZE	TEST SIZE	%POS
EEG	14	5000	9980	45%
OCC	5/6	5000	15560	23%
LEA	38	5000	4879	50%
CSG	103	5000	117410	49%

3 Experiment

From each data set, we randomly select 5000 training samples and designate leftover samples as test data. On each set of training data, we run 5-fold cross validation and measure validation set performance for each set of hyperparameters by the outlined metrics of ACC, FSC, and AUC. We select the hyperparameters with the best performance for each outlined metric and train algorithms on the entire training data for each set of best hyperparameters. We then measure test set performance with each algorithm by outlined metrics. We conduct five trials of this method and measure average performance over trials.

4 Results and Discussion

The results of the experiment are outlined in Tables 2 and 3, with additional results and p-values reported in the appendix tables. In each table, **bolded** values indicate the best performing algorithm and *s denote algorithms with non-significant differences to the best performing algorithm, where $p \geq .95$.

Table 2: Normalized scores of each learning algorithm on test set by metric (average over four data sets)

MODEL	ACC	F1	AUC	MEAN
LOG-REG	.7762	.7488	.7614	.7621
SVM	**.8514**	**.8479**	**.8517**	**.8503**
RF	.8347	.8247	.8333	.8309

Table 2 lists algorithm by row and average test set scores for each metric over four data sets and five trials, with a column for average performance over multiple metrics. Table 3 lists algorithm by row and average test set scores on each data set over five trials and three metrics, with a column for average performance over multiple data sets. The p-values for each metric in Tables 2 and 3 are listed in Tables 7 and 8 in Appendix. Raw test set values for each metric and data set are reported in Table 6 in Appendix. Additionally, training set performance by algorithm for each metric and data set is listed in Tables 4 and 5 in Appendix.

Table 3: Normalized scores of each learning algorithm on test set by data set (average over three metrics)

MODEL	EEG	OCC	LEA	CSG	MEAN
LOG-REG	.6048	.9854*	**.7319**	.7265	.7621
SVM	**.9343**	**.9857**	.7294	.7518	**.8503**
RF	.8610	.9834	.7208	**.7585**	.8309

Overall, the best classifier for each metric was SVM, followed by random forests and logistic regression. SVM has a clear performance advantage in all three metrics of ACC, F1, and AUC with no other algorithm having a statistically significant similarity in performance over metric. In average scores by data set, SVMs lead in the EEG and OCC data sets, while random forests score the highest on CSG, and logistic regression surprisingly scores the highest on LEA. The performance of logistic regression on LEA is likely due to LEA being slightly more linearly separable than the other data sets, despite having the second largest number of features. Logistic regression shows a statistically significant similarity in performance to SVMs on the OCC data set. This similarity is possibly due to the low number of features in OCC and an imbalance in labels, with positive labels consisting of only around 24%. This is supported by the high performance of all three algorithms on OCC. SVMs have a large statistically significant difference in performance over other algorithms in the EEG data set ($p < .05$). The overall performance of SVMs represent a surprising result as CNM06 (2006) showed a clear advantage of random forests in uncalibrated models.

Random forests presented superior performance on training data in all three metrics and the EEG, OCC, and CSG data sets. The lower test set result of random forests show that they likely do not generalize well on test data. This is possibly due to random forests overfitting on training data. SVMs presented only slightly lower performance on test data over training data, indicating acceptable generalizability. Logistic regression also demonstrates slightly lower performance on test data indicating generalizability, however, logistic regression had the lowest overall performance in both training and test sets. In contrast to CNM06, this data shows that uncalibrated SVMs perform and generalize better than random forests or logistic regression on multiple sets of data with a range of feature sizes.

This higher performance of SVMs is likely due to the improvement in computational implementation since SVM^{light} [Joachims, 1998], used in CNM06. SVM^{light} attempts to “shrink” the optimization problem which likely improved SVM training time in the computationally constrained environment of CNM06, however, modern CUDA-accelerated implementations, such as cuML [Raschka et al., 2020] and ThunderSVM [Wen et al., 2018] bypass these computation constraints and likely result in better classifier performance. SVM^{light} acknowledges the current approach of sequential minimal optimization used in cuML and ThunderSVM, noting that this approach would likely result in similar classification performance to SVM^{light}. However, the higher performance of SVMs over random forests and logistic regression suggests that the choice of using SVM^{light} could have affected results in CNM06. Since this comparison uses different data sets than CNM06, the higher performance of SVMs is not conclusive and requires further exploration.

5 CUDA-accelerated Training Time

This study explores and attempts to replicate the results of CNM06 [2006] with modern implementations of each classifier along with CUDA-acceleration on GPUs. These implementations allow much faster training than the CPU bound implementations in CNM06. While CPU training continues to be popular due to its ease of use in small-scale machine learning, GPU acceleration is quickly approaching similar usability. Currently, machine learning libraries such as TensorFlow [Abadi et al., 2015] and PyTorch [Paszke et al., 2019] require time consuming data preprocessing and troubleshooting. While these libraries provide an exponential decrease in training time, they do not demonstrate ease of use. Currently, sklearn [Pedregosa et al., 2011] is the most popular library for simple CPU machine learning. However, cuML[Raschka et al., 2020], from the open GPU data science toolkit Rapids [Team, 2018], provides the same functionality as sklearn, but can be used for large scale machine learning as well.

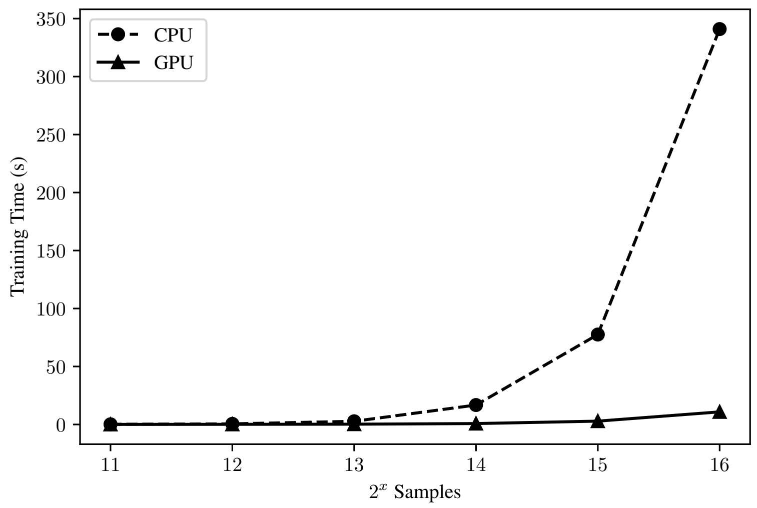

Figure 1: SVM training time on CPU and GPU by 2^x samples from CSG data set (average over five trials)

We examine the possible reduction in training time through using GPU acceleration on training through cuML over CPU training. We randomly select 2^{11} to 2^{16} training samples from our largest data set, CSG, and train each algorithm on each set of training samples. We measure training time as an average of five trials for each set of samples. We also train on the same set of samples for each algorithm by initiating a $random\_state$ value when sampling from our data set and setting each classifier in cuML to default sklearn values. We plot training time for each algorithm on CPU and GPU.

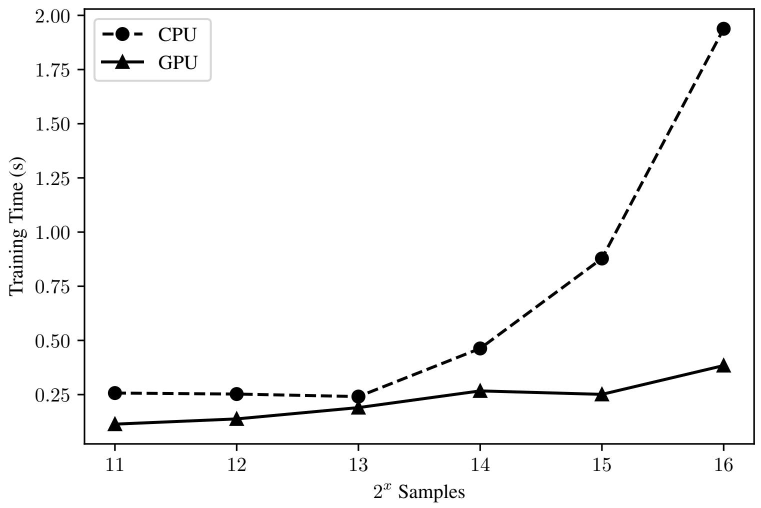

Figure 2: RF training time on CPU and GPU by 2^x samples from CSG data set (average over five trials)

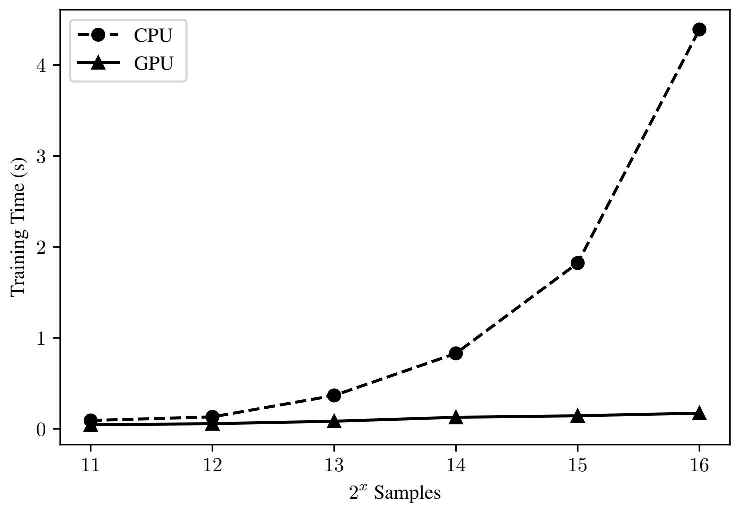

Figure 3: LOG-REG training time on CPU and GPU by 2^x samples from CSG data set (average over five trials)

In each algorithm, we can see that the training speed of CUDA-accelerated cuML greatly surpasses the training speed of CPU-based sklearn. For SVM and logistic regression, cuML training time increases linearly by a very small magnitude as training samples increase by 2^x. For random forests, cuML training time still increases linearly but at a slightly greater magnitude as training samples increase by 2^x. On the other hand, sklearn training time increases exponentially for each algorithm. This has major implications for large data sets, in which training time can reach unreasonable amounts for CPU-based sklearn. This is mainly seen with SVMs, where training time reaches multiple minutes on sklearn.

Overall, CUDA-accelerated cuML provides similar ease of use to CPU-based sklearn with most classifiers being drop-in replacements for sklearn. While sklearn classifiers are currently only used for education or small scale machine learning, cuML provides similar classifiers tuned towards large scale machine learning. cuML provides the tangible benefits of GPU acceleration through TensorFlow and PyTorch, without the complexity of learning each library and tuning data sets. As cuML naturally progresses, individuals who learn CPU machine learning through sklearn will be able to easily transfer their skills to GPU machine learning.

6 Appendix

Table 4: Normalized scores of each learning algorithm on training set by metric (average over four data sets)

MODEL	ACC	F1	AUC	MEAN
LOG-REG	.7794	.7513	.7637	.7648
SVM	.8675	.8663	.8701	.8680
RF	**.9874**	**.9835**	**.9870**	**.9860**

Table 5: Normalized scores of each learning algorithm on training set by data set (average over three metrics)

MODEL	EEG	OCC	LEA	CSG	MEAN
LOG-REG	.6041	.9860	.7314	.7377	.7648
SVM	.9747	**.9892**	.7330	.7750	.8680
RF	**.9793**	.9859	**.9996**	**.9790**	**.9860**

Table 6: Raw test set scores by data set

	EEG			OCC			LEA			CSG		
MODEL	ACC	F1	AUC	ACC	F1	AUC	ACC	F1	AUC	ACC	F1	AUC
	.9398	.9327	.9379	.9894	.9775	.9922	.7266	.7331	.7265	.7493	.7634	.7508
	.9352	.9273	.9340	.9893	.9771	.9882	.7235	.7234	.7260	.7498	.7599	.7497
SVM	.9387	.9319	.9375	.9886	.9758	.9913	.7256	.7217	.7263	.7505	.7517	.7509
	.9327	.9239	.9311	.9897	.9780	.9915	.7377	.7406	.7375	.7502	.7640	.7530
	.9413	.9351	.9359	.9891	.9768	.9916	.7327	.7263	.7337	.7478	.7375	.7480
	.8650	.8379	.8589	.9875	.9733	.9887	.7194	.7111	.7212	.7559	.7543	.7556
	.8752	.8502	.8677	.9886	.9758	.9913	.7299	.7273	.7301	.7609	.7675	.7618
RF	.8683	.8495	.8639	.9880	.9743	.9880	.7260	.7158	.7248	.7586	.7593	.7582
	.8701	.8406	.8602	.9870	.9726	.9884	.7149	.7094	.7140	.7531	.7594	.7544
	.8787	.8568	.8726	.9872	.9731	.9871	.7237	.7216	.7228	.7568	.7645	.7575
	.6448	.5500	.6304	.9888	.9761	.9905	.7301	.7294	.7304	.7451	.7452	.7494
	.6433	.5549	.6293	.9894	.9774	.9918	.7338	.7359	.7338	.7418	.7229	.7444
LOG-REG	.6295	.5358	.6149	.9886	.9760	.9910	.7262	.7281	.7262	.7452	.7399	.5000
	.6424	.5490	.6280	.9887	.9761	.9906	.7286	.7257	.7288	.7459	.7437	.7462
	.6383	.5556	.6253	.9888	.9759	.9907	.7403	.7412	.7403	.7439	.7379	.7460

Table 7: p-values for test set scores by metric

MODEL	ACC	F1	AUC
LOG-REG	.4779	.3916	.4150
SVM	1.000	1.000	1.000
RF	.8580	.7919	.8432

Table 8: p-values for test set scores by data set

MODEL	EEG	OCC	LEA	CSG
LOG-REG	.0003	.9552*	1.000	.0968
SVM	1.000	1.000	.0014	.0364
RF	.0006	.7378	.0041	1.000

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. *Proceedings of the 23rd international conference on Machine learning - ICML '06*, 2006:161–168, 06 2006. doi: 10.1145/1143844.1143865.

Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

T. Joachims. Making large-scale svm learning practical. LS8-Report 24, Universität Dortmund, LS VIII-Report, 1998.

R King, C. Feng, and A. Sutherl. Statlog: Comparison of classification algorithms on large real-world problems. *Applied Artificial Intelligence*, 9, 11 2000. doi: 10.1080/08839519508945477.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Sebastian Raschka, Joshua Patterson, and Corey Nolet. Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence. *arXiv preprint arXiv:2002.04803*, 2020.

Riot. League of Legends diamond ranked games (10 min), 2020. URL `https://www.kaggle.com/bobbyscience/league-of-legends-diamond-ranked-games-10-min`.

RAPIDS Development Team. *RAPIDS: Collection of Libraries for End to End GPU Data Science*, 2018. URL `https://rapids.ai`.

Valve and M. Machado. CS:GO professional matches, 2020. URL `https://www.kaggle.com/mateusdmachado/csgo-professional-matches`.

Zeyi Wen, Jiashuai Shi, Qinbin Li, Bingsheng He, and Jian Chen. ThunderSVM: A fast SVM library on GPUs and CPUs. *Journal of Machine Learning Research*, 19:797–801, 2018.