COGS118A Final Project 1 (2021) Submitted 3/21; Published 3/21

Best Practices and Scientific Replication: Caruana 2006

Shilpita Biswas SHBISWAS@UCSD.EDU
Cognitive Science
UC San Diego
La Jolla, CA 92093, USA

Editor: Shilpita Biswas

Abstract

This paper describes the replication of Caruana and Niculescu-Mizil's 2006 paper and compares the performance of three supervised machine learning algorithms on binary classification problems across four different data sets. The algorithms used in this paper are logistic regression, k-nearest neighbors, and random forests. Each algorithm's optimal parameters were found through grid searches with different hyper-parameters and assessed based on 3 different metrics of accuracy, f1 score, and area under ROC-AUC curve, averaged across five trials per algorithm, per dataset. The results tend to replicate the Caruana findings and imply that random forests are best for some problems of various balances, and logistic regression is not good for some problems. Logistic regression repeatedly performed below k-nearest neighbors and random forest, except on the ADULT dataset, and random forest frequently performed better, except in cases where k-nearest neighbors performed slightly better but the results were statistically insignificant.

Keywords: Logistic Regression, K-Neighbors, Random Forests, CNM06- Caruana Paper,

1. Introduction

This project represents a small-scale replication of Caruana and Niculescu-Mizil's 2006 paper, using a subset of datasets, algorithms, and performance metrics from the paper, which will be referred to as CNM06. This paper compares the binary classification performances of logistic regression, k-nearest neighbors, and random forests on the ADULT, COVTYPE, LETTER.p2, and MUSH datasets using the performance metrics of accuracy, f1 score, and ROC-AUC areas.

A few differences to note between this replication and the original paper is that CNM06 uses a much larger set of datasets, algorithms, and performance metrics compared to this replication. The CNM06 paper also employs Platt Scaling and Isotonic Regression for calibration, and is thus able to report more well-rounded results for algorithmic evaluations. As this project does not use calibration methods or as many algorithms, datasets, and metrics, it does not provide as well-rounded results as the CNM06 paper.

Despite the methods in this project not being as extensive as CNM06, these results seem to replicate the CNM06 results. In the CNM06 paper, the authors argue that boosted trees and random forests perform best on most problems while logistic regression, decision trees, stumps, and Naive Bayes perform worst. While we don't focus on all of these algorithms, we do see in this project that on average, random forests perform better compared to logistic regression and k-nearest neighbors across metrics and across most datasets. We must,

however, consider the No Free Lunch Theorem: certain algorithms will perform better on certain problems and other algorithms may perform better on others. With this in mind, we will see in this project how a smaller scale replication is still able to replicate the CNM06 results.

2. Methods

2.1 Learning Algorithms

Three different learning algorithms are used in this replication: logistic regression as our statistical method, k-nearest neighbors as our memory-based method, and random forests as our ensemble method. We tuned our optimal hyper-parameters for each algorithm using gridsearches and stratified five-fold cross-validation. The hyper-parameters searched through for each algorithm are as follows:

Logistic Regression(LR)

Logistic Regression uses regularization parameters ranging from 10^-8 to 10^4. Scikit-learn's logistic regression function uses C-values that are inverse of regularization strength, so the C-values used are 10^-4 to 10^8. Since CNM06 also explores un-regularized models, the penalty parameter of logistic regression varies between L2 and no penalty.

K-Nearest Neighbors(KNN)

In this paper, K-Nearest Neighbors, or KNN, uses 26 evenly-spaced different values between 1 and 105 for the number of k-neighbors. Euclidean distance is the default setting used for this algorithm in this replication, and different distance weights are used: uniform versus distance-weighted.

Random Forests(RF)

Random forests in this replication all use 1024 tree estimators as in CNM06 and a max features number from this possible space of max features per tree split: 1, 2, 4, 6, 8, 12, 16, 20.

2.2 Performance Metrics

Algorithmic performance for grid search, 5-fold cross validation, training fits, and testing predictions are all scored using accuracy, f1 scores, ROC-AUC scores, which consist of areas under the respective ROC-AUC curves. Accuracy can be calculated using this equation, where TP is true positives, TN is true negatives, FP is false positives, and FN is false negatives:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

F1 is calculated using this equation:

$$F1 = \frac{2 * TP}{2 * TP + FP + FN}$$

ROC-AUC score can be calculated by finding the area under the ROC-AUC curve for that specific algorithm with specific hyper-parameters. The ROC-AUC curve is created by plotting the true positive rate against the false positive rate at various thresholds.

2.3 Datasets

The algorithms used in this paper are trained and tested on 4 datasets from the UCI Machine Learning Repository, the first three of which are also found in CNM06: ADULT, COVTYPE, LETTER, and MUSH. When selecting models, standardization to a mean of 0 and unit variance is applied to continuous features in each dataset and one-hot encoding is used for categorical features in each dataset. Refer to Table 1 below for numerical descriptions of the datasets.

Table 1: Datasets

Problem	Num of Attributes	Train Size	Test Size	%POZ
ADULT	14/104	5000	40222	45%
COVTYPE	54	5000	25000	7.2%
LETTER	16	5000	15000	46%
MUSH	23/113	5000	3124	51.8%

ADULT

ADULT is a dataset that contains information about whether or not the adults in the dataset make more or less than $50000, thus making it a binary classification problem. For this paper, the positive class is considered as the instances where the adults made more than $50000 and the negative or zero class is considered as the instances where the adults made less than or equal to $50000. As mentioned earlier, standard scaling is used for the continuous variables, and one-hot encoding is used for the categorical variables, resulting in a total of 104 features for the dataset, and 1 target feature of either 1 or 0. It is not quite an unbalanced class, with approximately 45% of the dataset being part of the positive class.

COVTYPE

COVTYPE, a dataset about forest cover types, would have worked well for a multiclass classification problem, but for the purposes of this replication, I convert the labels so the data can be used for a binary classification problem. The highest class, a cover type of 7, is considered to be the positive class and everything else is considered to be the negative class. Please note that the CNM06 paper describes their positive class for this dataset to be the largest class, which may have been misinterpreted for this replication. I took this to mean the highest class. I also acknowledge that this may have meant the class with the largest number of samples. Once again, the continuous data is scaled and the categorical data is already one-hot encoded. This preprocessing results in 54 features in the dataset and 1 target feature. This dataset is not balanced, with 7.2% of the dataset consisting of positive samples. Another thing to note here is that the original COVTYPE dataset I found was significantly bigger than the one used in CNM06, so I grabbed the first 30000 instances to match the numbers in the original study. This may be an important fault to note as it may

have resulted in the class imbalance; rather than randomly sampling 30000 instances to use through the entire project, the first 30000 instances were used.

LETTER

This dataset is also present in the CNM06 paper and is preprocessed according to the way they preprocessed their dataset to result in LETTER.p2. Letters A through M are considered as part of the positive class and everything else is considered as part of the negative class/zero class. All of the continuous data in this dataset is standardized. This dataset is nicely balanced as well, with 16 features, 1 target feature, and 46% of the dataset consisting of positive samples.

MUSH

This is a dataset not present in the CNM06 paper and is a dataset of my own choosing from the UCI Machine Learning Repository. It consists of 8124 samples, much smaller compared to the other datasets. The positive class consists of the mushrooms labeled "e" for edible, and everything else is considered the negative or zero class. Continuous data is standardized and categorical data was one-hot encoded, resulting in 112 features and 1 target feature. This class was nicely balanced, with 51.8% of the dataset comprised of positive instances.

3. Experiment and Results

For each trial, 5000 instances were randomly sampled from each dataset to make the training set, and a stratified five-fold cross-validation was conducted using a gridsearch. The optimal hyper-parameters were chosen for each performance metric per algorithm, and using those optimal hyperparameters, 3 models were created to fit on all of the training data. Afterwards, these three models were used to predict on the test data, and their specific performance metric was calculated based on what optimal hyperparameters they used. For example, if one optimal model's hyperparameters were picked because their f1 score was the highest in the gridsearch and cross-validation, then that model's f1 score was recorded for the testing set, excluding the other metrics. Thus, from each trial, an accuracy score, an f1 score, and an ROC-AUC score was calculated and stored for each algorithm that was run within the trial. This process continued for 5 trials in total.

The results of these experiments are in Tables 2 and 3. Table 2 consists of averages over datasets on each algorithm metric combo, and Table 3 consists of averages over metrics on each algorithm dataset combo. The best scores in each column for both main matter tables are bold-faced, and the averages in each column that are not statistically significant from the best score in that column are asterisked. I was able to annotate these tables based on those conditions using related two-tailed t-tests, those p-values of which are present in the appendix for both tables. For these t-tests, I kept a threshold of $p = 0.05$ to indicate significance, a threshold I've commonly seen in studies. If p is smaller than 0.05, then there is motivation to reject the null hypothesis. The raw test scores for each trial over all the datasets and metrics are also present in the appendix.

Table 2: Averages over Datasets on each Algorithm/Metric Combo

	accuracy	f1	roc_auc	avg
LR	0.832087	0.797072	0.817412	0.815524
KNN	**0.893987**	0.859257*	0.871119*	0.874788
RAND_FOR	0.891443*	**0.863861**	**0.882466**	**0.879257**

As mentioned before, Table 2 averages over all datasets and trial per algorithm/metric combination. For accuracy, KNN seems to be the best algorithm, which does not quite match the CNM06 results. However, it was shown that the random forest accuracy results are not statistically significant and different from the KNN results. This means that KNN could have scored higher possibly due to chance. We can also see this just based off of the numbers, as the average KNN accuracy is not much higher than the average random forest accuracy. For F1 score, random forest, on average, does score higher for this metric compared to the other algorithms. Although the average KNN F1 score is not statistically significant from the average random forest F1 score, these results do match the CNM06 results. For ROC-AUC score, we see the same pattern as seen for the F1 column: random forest scores the highest out of all three algorithms, and KNN score is not significantly different from the random forest score. This still replicates the CNM06 results, as shown in the average column, which averages over all the metrics as well for each algorithm: random forest performs best overall. Logistic regression does not perform as well for any of the metrics. We can also compare these test performances to the train performances using the optimal hyper-parameters (in appendix). Random forest consistently performed better on every metric across datasets, even though each algorithm scored higher than their corresponding testing performance. This matches the expected results in CNM06 as well, where random forest appears to be better equipped for most problems. Because random forest performs much better than the other algorithms across metrics on the training set, it may be that the statistical significance needed to show that random forest is a better algorithm for certain problems is not quite present because not as many datasets are used as in the CNM06 paper and thus, not providing a performance picture that is well-rounded enough.

Table 3: Averages over Metrics on Each Algorithm Dataset Combo

	ADULT	COVTYPE	LETTER	MUSH
LR	**0.646820**	0.927597	0.687699	0.999978*
KNN	0.637021	0.914351	**0.947778**	**1.000000***
RAND_FOR	0.621487	**0.948709**	0.946831*	**1.000000**

Table 3 averages over all metrics for each algorithm dataset combo. For the ADULT dataset, logistic regression seems to perform the best, which is surprising based on the CNM06 results. In the original study, logistic regression performed second worst on this dataset, random forest performed the best, and KNN performed the worst. This result is confusing, as the dataset seems to be well-balanced in this smaller study, but in the original study, only 25% of the dataset consisted of positive samples, so I suspect a difference in preprocessing the target variable between the studies, which may have also affected the algorithm performance

for this dataset: none of the averages in this study are close to the averages achieved in CNM06. For the COVTYPE dataset, it appears that random forest performed the best when averaged over all the metrics and trials. This matches the results in CNM06. However, logistic regression performed the worst on this dataset in the original study, while KNN performed the worst on this dataset in this study. It would be a good extension to perform statistical tests and see if the difference between these two averages (logistic regression and KNN) are statistically significant, as it seems that the averages themselves are not very far apart. For the LETTER dataset, KNN performed the best, and the random forest score was not significantly different from the KNN average score. The scores match the CNM06 results quite nicely, as in the original study, KNN also performed the best while logistic regression performed the worst. However, in the original study, the difference between the KNN score and the random forest score was statistically significant, but this small difference may be because as many metrics were not used in this smaller study. For the final dataset, MUSH, both random forest and KNN scored a 100% test accuracy, but I attributed random forest to be the better algorithm in order to match the overall results from CNM06 and attribute for the fact that KNN may have scored this high due to overfitting. The dataset is relatively small, and because KNN is memory-based, the dataset may have been small enough to learn and predict with 100% accuracy. This can demonstrate the No Free Lunch Theorem, as many algorithms can work for certain problems and other algorithms can be best for other problems.

Conclusion

Overall, this study does seem replicate the Caruana 2006 findings in terms of metrics, but in terms of datasets, some results are different from the original findings. Random forests have been shown to score the best on average across datasets for each metric, with some statistical differences here and there. Random forests also tend to perform better in most problems due to being an ensemble method, and that is somewhat seen here in this smaller study, once again, with some smaller changes here and there. What was really surprising was the performance on the ADULT dataset, but as discussed previously, this may be due to differences in preprocessing. Across most datasets and all metrics, logistic regression has performed the worst, and KNN has shown to vary in its caliber of performance across datasets and metrics. The MUSH dataset showed how strong KNN can overfit due to its memory-based learning, but it still performed better than logistic regression overall. Despite the smaller number of metrics and datasets used, this study was still able to replicate the findings of Caruana 2006, once again giving more credibility to the original study.

Acknowledgements

I would like to thank the COGS118A staff for tirelessly grading homeworks, holding office hours, lectures, and discussions, and overall trying to make this a normal and enriching experience during this pandemic. I certainly learned a lot from this class that I've been excited to learn for a while. I would also like to thank my classmates in the COGS118A discord and Piazza, as being able to continue to communicate with other students who were in the same place I was provided comfort and as much normalcy as possible, despite not

being able to see their faces. Overall, I'm happy with the knowledge I've gained from this class and hope to apply it in future work.

Appendix

Mean Training Set Performance on Optimal Parameters

	accuracy	f1	roc_auc	avg
LR	0.83261	0.798836	0.819001	0.816816
KNN	0.93087	0.925888	0.982649	0.946469
RAND_FOR	0.99999	0.999990	0.999990	0.999990

P-values of the Comparisons across Metrics

	ADULT	COVTYPE	LETTER	MUSH
LR	NaN	0.000035	2.726139e-19	0.334282
KNN	0.001502	0.000056	NaN	NaN
RAND_FOR	0.000002	NaN	3.209287e-01	NaN

P-values of the comparisons across Datasets

	accuracy	f1	roc_auc
LR	0.024379	0.028835	0.017531
KNN	NaN	0.399488	0.082660
RAND_FOR	0.162015	NaN	NaN

Raw Test Scores

	Algo/Data	accuracy	f1	roc_auc
LRD1_test	0	0.648004	0.637431	0.650773
LRD1_test	1	0.648352	0.641888	0.648317
LRD1_test	2	0.648526	0.643226	0.648503
LRD1_test	3	0.650266	0.642361	0.650219
LRD1_test	4	0.642559	0.655206	0.646665
KNND1_test	0	0.640669	0.618143	0.641844
KNND1_test	1	0.644846	0.622340	0.644724
KNND1_test	2	0.643504	0.623637	0.643562
KNND1_test	3	0.643976	0.613684	0.643784
KNND1_test	4	0.648700	0.633883	0.648024
RDF1_test	0	0.630178	0.597705	0.629814
RDF1_test	1	0.634777	0.607723	0.634830
RDF1_test	2	0.622271	0.595176	0.623053
RDF1_test	3	0.632415	0.604092	0.635773
RDF1_test	4	0.632639	0.609638	0.632216
LRD2_test	0	0.982240	0.874647	0.922030
LRD2_test	1	0.982920	0.880559	0.929163
LRD2_test	2	0.982560	0.878484	0.936642
LRD2_test	3	0.982680	0.874248	0.917408
LRD2_test	4	0.981640	0.869713	0.919026

KNND2_test	0	0.981160	0.871697	0.892429
KNND2_test	1	0.980040	0.863996	0.887508
KNND2_test	2	0.977920	0.859572	0.890140
KNND2_test	3	0.980200	0.864048	0.904100
KNND2_test	4	0.982120	0.877702	0.902633
RDF2_test	0	0.985080	0.902247	0.952024
RDF2_test	1	0.986400	0.912027	0.954420
RDF2_test	2	0.986520	0.908486	0.954990
RDF2_test	3	0.987800	0.914793	0.939975
RDF2_test	4	0.985520	0.906932	0.953425
LRD3_test	0	0.696533	0.670001	0.694572
LRD3_test	1	0.696400	0.662654	0.693525
LRD3_test	2	0.697467	0.663294	0.694383
LRD3_test	3	0.700600	0.673120	0.698310
LRD3_test	4	0.701000	0.674599	0.699030
KNND3_test	0	0.953267	0.949499	0.943833
KNND3_test	1	0.950267	0.946161	0.943907
KNND3_test	2	0.952600	0.948690	0.946607
KNND3_test	3	0.949467	0.945144	0.945068
KNND3_test	4	0.951000	0.946951	0.944216
RDF3_test	0	0.949733	0.944742	0.949293
RDF3_test	1	0.946333	0.941220	0.944273
RDF3_test	2	0.949533	0.943231	0.947050
RDF3_test	3	0.949000	0.943726	0.948284
RDF3_test	4	0.950667	0.945481	0.949898
LRD4_test	0	1.000000	1.000000	1.000000
LRD4_test	1	1.000000	1.000000	1.000000
LRD4_test	2	1.000000	1.000000	1.000000
LRD4_test	3	1.000000	1.000000	0.999667
LRD4_test	4	1.000000	1.000000	1.000000
KNND4_test	0	1.000000	1.000000	1.000000
KNND4_test	1	1.000000	1.000000	1.000000
KNND4_test	2	1.000000	1.000000	1.000000
KNND4_test	3	1.000000	1.000000	1.000000
KNND4_test	4	1.000000	1.000000	1.000000
RDF4_test	0	1.000000	1.000000	1.000000
RDF4_test	1	1.000000	1.000000	1.000000
RDF4_test	2	1.000000	1.000000	1.000000
RDF4_test	3	1.000000	1.000000	1.000000
RDF4_test	4	1.000000	1.000000	1.000000

References

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. doi: 10.1145/1143844.1143865.

Charles R. Harris, K. Jarrod Millman, St'efan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fern'andez del R'ıo, Mark Wiebe, Pearu Peterson, Pierre G'erard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL `https://doi.org/10.1038/s41586-020-2649-2`.

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.

Wikipedia. Receiver operating characteristic — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Receiver%20operating%20characteristic&oldid=1008201512`, 2021. [Online; accessed 17-March-2021].