War of The Algos: Algorithm Performance on Binary Classification Problems

Abstract

This paper evaluates the performance of three supervised machine learning algorithms that will tackle binary classification problems across four different datasets. The structure of the exploration is inspired by Caruana and Niculesu Mizil's paper, which also evaluates algorithms on binary classification tasks but on a larger scale. The algorithms discussed in this paper are random forest, k-nearest neighbors and logistic regression. Each dataset is assessed on all three algorithms, each algorithm is evaluated by three error metrics. The paper corresponded with CNM06's findings on the best performing algorithm.

Keywords: binary classification, random forest, knn, k-nearest neighbors, logistic regression, algorithm performance, supervised learning, supervised machine learning algorithms.

1. Introduction

In 2006, Caruana and Niculesu Mizil (2006) wrote a paper evaluating eight supervised machine learning algorithms on eleven different datasets. They found that random forests and boosted trees have the best performance, while logistic regression, decision trees, stumps, and naive bayes perform the worst. I decided to test their results by conducting a study of my own, and will be referring to their work through this paper as CNM06 (2006). I picked three algorithms from across the spectrum of CNM06's evaluation: one "best" algorithm (random forest), one "worst" algorithm (logistic regression) and one "average" algorithm (k-nearest neighbors). Each algorithm will be evaluated on three error metrics, and on four datasets.

2. Methodology

2.1 Learning Algorithms

There is a space of hyperparameters curated for each algorithm that will be searched through in order to find the best set of parameters according to the score of each error metric. In order to perform this search, I will be using sklearn's method gridsearchcv(). This method is also used to perform 5-fold cross validation on the training dataset in order to avoid overfitting, while searching for the best set of hyperparameters. The parameter space curated for each algorithm are as follows:

- **Random Forest:** As CNM06 does, the random forest algorithm will have 1024 trees. The max features to be considered at each split is chosen from this list: [1, 2 , 4, 6, 8, 12, 16 , 20], starting 1 and possibly going up till 20 depending on the number of features in the dataset.
- **kNN:** CNM06 uses 26 values ranging from 1 to the size of the training set for values of K. However, given time limitations, I used 26 values ranging from K = 1 to K = |trainset/10|. This is also due to the fact that the performance of kNN doesn't change much for high values of K, and the optimal value of K is usually found at the square root of the size of the dataset. Since the size of all my training sets was 5000, 500 seemed a safe number for K that would optimize both efficiency and performance. The distance metrics considered were Euclidean and Manhattan, and the weight functions considered for prediction were uniform and distance.
- **Logistic Regression:** Similar to CNM06, I considered both regularized and unregularized models, as well as values of 10^{-8} to 10_4 for the ridge parameter.

2.2 Error Metrics

The performance metrics I chose to use are:

- **Accuracy**: This is a basic metric commonly used to score the performance of algorithms. While accuracy can perform poorly on imbalance datasets, it is a good evaluation tool for balanced datasets. None of my training sets were imbalanced. It describes how many correct predictions out of the total number of predictions that were made.

- **F1:** This metric is included to account for the algorithm's performance in terms of both the false negative rate and false positive rate. It gets the best precision and recall at the same time.
- **AUC:** This metric is included in order to evaluate the algorithm's performance across all the possible classification thresholds.

2.3 Datasets

Each dataset with nominal columns was one hot encoded, and any continuous columns were standardized in order to ensure one feature wouldn't have more weight on the classification than the other. Table 1 shows the number of attributes, the train set size, the test set size, and the percentage of the positive class in each dataset.

- **ADULT**: On this dataset we are classifying whether an individual makes above or below mean salary based on their background. Those who make above mean salary were put in the positive class, and those who make below mean salary were put in the negative class. This dataset was imbalanced, but I was able to balance the training sets by random resampling. Columns and rows with missing/NaN values were dropped.
- **ENERGY:** On this dataset we are classifying whether a power plant has above or below mean electrical energy output per hour based on various ambient conditions. The target column was initially continuous so I calculated the mean and transformed the column: energy outputs that were above the mean were put in the positive class and vice versa. Columns and rows with missing/NaN values were dropped.
- **OCCUPANCY:** On this dataset we are classifying whether or not a room will be occupied based on conditions inside the room. An occupied room was placed in the positive class and vice versa. Columns and rows with missing/NaN values were dropped.
- **MUSHROOM:** On this dataset we are classifying whether or not a mushroom will be edible or poisonous depending on its physical attributes. An edible mushroom was placed in the positive class and vice versa. Columns and rows with missing/NaN values were dropped

Table 1: Description of Datasets

PROBLEM	#ATTR	TRAIN SIZE	TEST SIZE	%POZ
ADULT	105	5000	27560	75%
OCCUPANCY	6	5000	5145	47%
MUSHROOM	116	5000	3123	51%
ENERGY	4	5000	4568	46%

3. Experiment and Results

3.1 Performance By Metric

The following structure of training and obtaining parameters was taken from CNM06. Each dataset was divided into training and testing sets 5 times through random resampling. I then used each training set to train a model and obtain the best set of hyperparameters for each metric. In a single trial, each training set went through 5-fold cross validation, meaning that 4000 out of 5000 data samples were used to train a model, and the remaining 1000 samples were used for validation. This process occurred for each training set five times. The evaluation for this validation was done using accuracy, F1 and AUC scores. For each training set, I obtained three ideal sets of parameters based on the three evaluation metrics. For each training set (5 sets), I trained three models using these three sets of parameters, and generated predictions based on both the training and test set. Finally, I evaluated the correctness of these predicted values by comparing them with the true values. Table 2 displays the performance of the metrics with each algorithm.

Table 2: Mean Test Performance Across Metrics

	ACC	ROC	F1	MEAN
KNN	0.948*	0.918	0.962	0.942
LOGREG	0.949	0.926	0.962	0.945
RANDOM FOREST	**0.953**	**0.932**	**0.965**	**0.950**

The algorithm that had the best score by metric is marked in boldface. I conducted ttests to explore whether the performance of the other algorithms was not statistically distinguishable compared to the best algorithm, but found that all of the algorithms were statistically distinguishable, with p values < 0.05 on paired ttests. I reconducted the ttests with p = 0.01, to take into account the number of trials performed. This added one star to the table, but the ideal way to confirm the statistical significance differences between the algorithms would be to conduct more trials.

Random Forest performed the best when comparing by metric, with Logistic Regression performing at second place and kNN at third. However, announcing one algorithm as the best or the worst would be a stretch. Each algorithm actually seems to perform similarly to each other. Upon comparing the test set performances to train set performances (table 4, appendix) one can see that kNN and Random Forest had close to perfect performance, whereas Logistic Regression did noticeably worse than the previous two. The performance of Logistic Regression from train set to test set stayed about the same, the train set performing only slightly better.

3.2 Performance By Problem

Similarly to its performance by metric, Random Forest uniformly outperforms the other two algorithms. I had a few opportunities to run trials several times, and found that random forest was the “safest” and performed consistently. This is also echoed in the raw test set scores seen in the appendix. kNN was consistent but not as much as Random Forest. Logistic Regression had the most consistent low scores, but it was also the one whose performance varied for every metric and across trials.

Table 3: Mean Test Performance Across Datasets

	ADULT	OCCUPANCY	MUSHROOM	ENERGY	MEAN
KNN	0.8167	0.991	1.0	0.9617	0.942
LOGREG	0.8359	0.991	1.0	0.9566	0.946
RANDOM FOREST	**0.8414**	**0.992**	**1.0**	**0.9653**	**0.950**

3.3 Analysis of Time Complexity and Parameters

The time complexity for Logistic Regression is o(d), and it was the fastest algorithm to run out of all three. Examination of the gridsearch object after fitting logistic regression models shows me logistic regression has the smallest mean fit times. While the preferred parameters for C and Regularization model had a big range of values, I found that the consistently preferred solver for this algorithm was saga. This may just be because the saga solver is flexible.

The time complexity for kNN O(d * n * log(n)), and this was the slowest algorithm to run out of all three. I found that as K increases, the mean fit time for a model stays about the same, but the amount of time it takes to score the model increases significantly. Since I went up to almost 500 neighbors in all trials, and had three scoring metrics, this reason would be partially responsible for the longer runtimes. The mushroom dataset, which received perfect scores, had its best parameters set at K=1. The other datasets had their best parameters at anywhere from $1 \leq K \leq 100$. The optimal value of K is often found at the square root of the size of the training set, which in this case is ~70. My findings seem to fit in with that.

The time complexity of Random Forest is O(n*log(n)*d*k), where k is the number of trees. Although usually this algorithm is pretty fast, it was slightly slower in this case given that the number of trees was 1024 for all problems. Interestingly, when I checked the max mean fit time for all algorithms, I found random forest to have the largest values. However, it did not take as much time as kNN. It seems that for a singular model, random forest takes the largest amount of time to fit. However, perhaps it takes less total time to run because there are lesser models to fit or a smaller number of parameters to check. I confirmed this idea after converting the gridsearch object into a dataframe. Even for the dataset with the most features, random forest has at most 8 combinations of parameters to check, whereas knn has more than 100.

4. Conclusion

In conclusion, my findings echo that of CNM06, implying Random Forest is the best classifier. CNM06 did not name Random Forest as the very best, calling other algorithms safer even if they do not perform better. However, there may be some reasons that my findings are not robust. Given the high scores achieved by all algorithms, it is possible that all the datasets I chose had attributes directly related to the label, or highly correlated to the label. For further exploration, I should look for datasets that do not historically produce such high scores, and test them on these algorithms and more. I do not think that the high scores are because of sampling problems. I checked the training class distribution for every training set for every problem, and made sure none were imbalanced, even if the dataset itself was imbalanced. Further explorations should also include more error metrics, and a larger number of trials. Another problem that could have occurred is that I did not use calibration in my process. This could have impacted the scores. All in all, my findings showed me the scope this study holds when done on a larger scale.

Acknowledgements

I would like to acknowledge Professor Fleischer, and Abdullah and Enoch for all the help they offered in the duration of COGS 118A that afforded me the knowledge to carry out this project. I would also like to acknowledge CNM06 for being the pioneer of this structure of exploration, which allowed me to conduct studies in a similar manner and learn.

Bonus

I would like to be considered for extra credit because I spent a significant amount of time searching for datasets on the UCI repository, downloading many of them to check whether they are imbalanced or not. For several datasets that were not already formatted for binary classification, I also did subject matter research. For instance, when I was considering studying the dry beans dataset, I spent some time researching the 7 types of beans featured in the dataset to decide whether I could put them in two classes. I ultimately decided against it.

Another reason I think I can be considered for extra credit is because I attempted an analysis of the gridsearch objects for the models to learn about how much time they took, how gridsearch actually searches and what parameters were the best ones. Thank you.

Appendix

NOTE: CODE ATTACHED AFTER REFERENCES

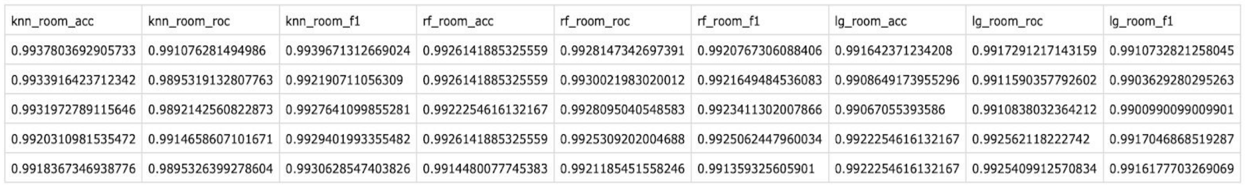
Table 4: Mean Training Performance Across Metrics

	ACC	ROC	F1	MEAN
KNN	**0.999**	**0.999**	0.999	0.999
LOGREG	0.950	0.928	0.963	0.947
RANDOM FOREST	**0.999**	**0.999**	**0.9997**	**0.9992**

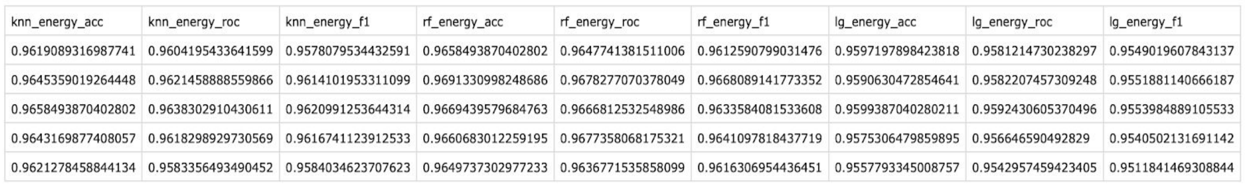
Table 5: P-Value For Table 2

	P-Value LogReg	P-Value kNN
RF ACCURACY	0.002	0.012
RF ROC	0.0006	0.007
RF F1	0.003	0.007

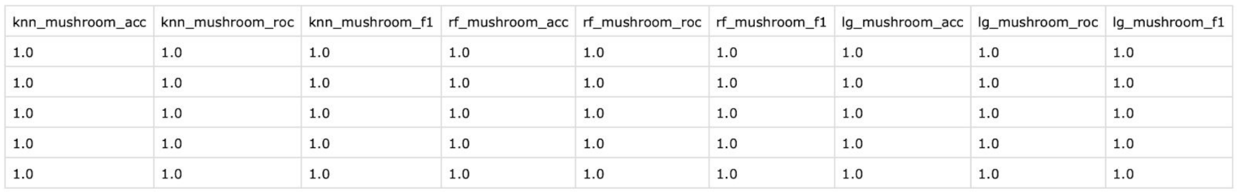
Raw test scores for all datasets



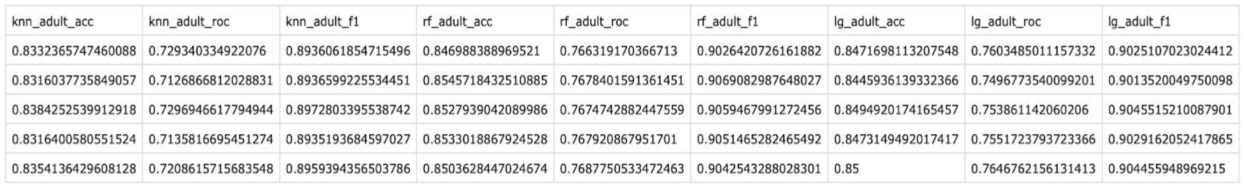
knn_room_acc	knn_room_roc	knn_room_f1	rf_room_acc	rf_room_roc	rf_room_f1	lg_room_acc	lg_room_roc	lg_room_f1
0.9937803692905733	0.991076281494986	0.9939671312669024	0.9926141885325559	0.9928147342697391	0.9920767306088406	0.991642371234208	0.9917291217143159	0.9910732821258045
0.9933916423712342	0.9895319132807763	0.992190711056309	0.9926141885325559	0.9930021983020012	0.9921649484536083	0.9908649173955296	0.9911590357792602	0.9903629280295263
0.9931972789115646	0.9892142560822873	0.9927641099855281	0.9922254616132167	0.9928095040548583	0.9923411302007866	0.99067055393586	0.9910838032364212	0.9900990099009901
0.9920310981535472	0.9914658607101671	0.9929401993355482	0.9926141885325559	0.9925309202004688	0.9925062447960034	0.9922254616132167	0.992562118222742	0.9917046868519287
0.9918367346938776	0.9895326399278604	0.9930628547403826	0.9914480077745383	0.9921185451558246	0.991359325605901	0.9922254616132167	0.9925409912570834	0.9916177703269069



knn_energy_acc	knn_energy_roc	knn_energy_f1	rf_energy_acc	rf_energy_roc	rf_energy_f1	lg_energy_acc	lg_energy_roc	lg_energy_f1
0.9619089316987741	0.9604195433641599	0.9578079534432591	0.9658493870402802	0.9647741381511006	0.9612590799031476	0.9597197898423818	0.9581214730238297	0.9549019607843137
0.9645359019264448	0.9621458888559866	0.9614101953311099	0.9691330998248686	0.9678277070378049	0.9668089141773352	0.9590630472854641	0.9582207457309248	0.9551881140666187
0.9658493870402802	0.9638302910430611	0.9620991253644314	0.9669439579684763	0.9666812532548986	0.9633584081533608	0.9599387040280211	0.9592430605370496	0.9553984889105533
0.9643169877408057	0.9618298929730569	0.9616741123912533	0.9660683012259195	0.9677358068175321	0.9641097818437719	0.9575306479859895	0.956646590492829	0.9540502131691142
0.9621278458844134	0.9583356493490452	0.9584034623707623	0.9649737302977233	0.9636771535858099	0.9616306954436451	0.9557793345008757	0.9542957459423405	0.9511841469308844



knn_mushroom_acc	knn_mushroom_roc	knn_mushroom_f1	rf_mushroom_acc	rf_mushroom_roc	rf_mushroom_f1	lg_mushroom_acc	lg_mushroom_roc	lg_mushroom_f1
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0



knn_adult_acc	knn_adult_roc	knn_adult_f1	rf_adult_acc	rf_adult_roc	rf_adult_f1	lg_adult_acc	lg_adult_roc	lg_adult_f1
0.8332365747460088	0.729340334922076	0.8936061854715496	0.846988388969521	0.766319170366713	0.9026420726161882	0.8471698113207548	0.7603485011157332	0.9025107023024412
0.8316037735849057	0.7126866812028831	0.8936599225534451	0.8545718432510885	0.7678401591361451	0.9069082987648027	0.8445936139332366	0.7496773540099201	0.9013520049750098
0.8384252539912918	0.7296946617794944	0.8972803395538742	0.8527939042089986	0.7674742882447559	0.9059467991272456	0.8494920174165457	0.753861142060206	0.9045515210087901
0.8316400580551524	0.7135816695451274	0.8935193684597027	0.8533018867924528	0.767920867951701	0.9051465282465492	0.8473149492017417	0.7551723793723366	0.9029162052417865
0.8354136429608128	0.7208615715683548	0.8959394356503786	0.8503628447024674	0.7687750533472463	0.9042543288028301	0.85	0.7646762156131413	0.904455948969215

References

R. Caruana and A. Niculescu-Mizil. ”An empirical comparison of supervised learning al-gorithms.”In Proceedings of the 23rd international conference on Machine learning,161-168. 2006.

https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

https://scikit-learn.org/stable/modules/model_evaluation.html

https://machinelearningmastery.com/tour-of-evaluation-metrics-for-imbalanced-classification/

https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234