A Revised Empirical Comparison of Supervised Learning Algorithms

Alex Liebscher
Department of Cognitive Science
University of California: San Diego

Abstract

For any classification problem, choosing a proper classifier and its parameters is critical for success. To build on the existing push for methodical supervised machine learning, this work evaluates the performance of seven classification algorithms across four data sets. For thoroughness, each classifier was tested over three independent trials, where each trial was subject to three partitions of the data. For each partition of the data, cross validation was performed. For each CV fold, an optimal set of hyper-parameters was found for the classifier using Bayesian Search. Contrary to traditional grid search, this method improves performance, takes advantage of the underlying parameter space with specific priors, and reduces redundant and insignificant searches. Performance overall proved that Random Forests, Gradient Boosted Trees, and RBF-SVM achieve the highest results. K-Nearest Neighbors may also be a viable solution but should be treated with care and precision.

Introduction

For any classification problem, choosing a proper classifier and its parameters is critical for success. This is considered by many to be more of an art than a science. However, some previous work has attempted to introduce methodology to the practice and standardize beliefs about classification. (Caruana and Niculescu-Mizil 2006) (CNM), the main inspiration for this paper, tuned and evaluated the performance of many classifiers over many data sets to quantitatively assess and rank the performance of these classifiers. CNM's work heavily influenced the directions of this paper. Here, a set of classifiers is also evaluated on a variety of data sets. The average classification performance for seven classifiers over four data sets is determined and consequently the relative ranking of each classifier. It is undeniable that some of the seven classifiers will perform better over some data sets, thus four sets of data were tested on. Each classifier is tested over three trials of three data partitions. For each partition, the classifiers were cross validated and the hyper-parameters were optimized with Bayesian Search. In the end, the average performance on the testing set is reported and compared.

Related Work

CNM evaluated the performance of 10 classifiers on 11 data sets. They also considered 8 evaluation metrics and after calibrating each to represent a similar scale, averaged them and reported these scores for each classifier. Calibration of these metrics allowed them to not only rank the classifiers by performance ability, but also to perform a bootstrapped analysis. This analysis presented the likelihood that classifiers would be ranked in the way that they were originally reported. Overall, CNM is an excellent benchmark for most analyses of classification algorithms due to its thoroughness.

Yet, very few meta-analyses of this type have been performed. As CNM point out, one of the most significant publications of this type was the STATLOG study (King, Feng, and Sutherland 1995). King, Feng, and Sutherland performed a comprehensive analysis of then-state-of-the-art classification algorithms evaluated over 12 real-world data sets. In addition, (Ali and Smith 2006) explored and analyzed the performance of 8 classifiers over 100 data sets.

These studies influenced others with similar goals, albeit with more specificity regarding the data or application. For example, (Ahmed et al. 2010) comprehensively evaluated the performance of various regression algorithms over a single time-series data set. Other similar studies exist, however many of them are meta-reviews of supervised learning algorithms, such as (Kotsiantis, Zaharakis, and Pintelas 2007).

These studies have guided the direction of this work by providing material to found expectations on, and to build the methodology off of. Unsurprisingly, there is still much domain and machine learning knowledge required to even properly set up which classifiers to examine and the hyper-parameter spaces to evaluate over. Meta-learning, the idea that a system learns not only the hyper-parameters, but also the best classifier for the data, is widely debated and still an active area of research (Lemke, Budka, and Gabrys 2015).

Methods

Classification Algorithms

Seven classification algorithms were tested on the data. These include: Gradient Boosted Trees, Adaptive Boosting, Linear Support Vector Machine (SVM), Radial Basis Function SVM, Random Forest, K-Nearest Neighbors, and Logistic Regression. The parameter space is explored with Bayesian optimization methods, as described in the respective section below. Parameters and their ranges were modeled after CNM, however ranges were enlarged when reasonable due to the effectiveness of Bayesian optimization. For each data set, three trials were run. For each trial,

the data are shuffled and partitioned into training and testing sets. The first partition saves 80% of the data for testing, the second, 50%, and the third, 20%. At this point, hyper-parameters were then tuned with Bayesian Search over the training set, where 5-fold cross validation is performed with each hyper-parameter initialization. To further improve performance, each trial takes advantage of aggressive multiprocessing. After training, the best estimator, based on the best F1-Score from cross validation, is used to score the testing set.

Below are the specifics and the parameter spaces searched over for each classifier.

Gradient Boosted Trees (GBT) From the optimized Python package `xgboost` (Chen and Guestrin 2016). Using `gbtree`, a decision tree based model similar to CART used by CNM. Parameter space searched over: number of estimators (integer values between 2 and 1024). Up to 20 search iterations per parameter.

Adaptive Boosting (AdaBoost) Parameter space searched over: number of estimators (integer values between 2 and 1024), learning rate (log-uniform distribution between 10^{-7} and 10^{2}). Up to 20 search iterations per parameter.

Linear SVM (Lin-SVM) Parameter space searched over: regularization of the error term (log-uniform distribution between 10^{-8} and 10^{3}). Up to 20 search iterations per parameter.

Radial Basis Function SVM (RBF-SVM) Parameter space searched over: kernel regularization coefficient (log-uniform distribution between 10^{-4} and 2), regularization of the error term (log-uniform distribution between 10^{-8} and 10^{3}). Up to 12 search iterations per parameter.

Random Forest (RF) Number of estimators is 1024, using the entropy criteria for branching. Parameter space searched over: number of max features (integer values between 2 and the number of features). Up to 24 search iterations per parameter.

K-Nearest Neighbors (KNN) Parameter space searched over: weighting either uniform (simply majority vote of k nearest labels) or distance (points from x are weighted as $1/d$ where d is the Euclidean distance from the point x), number of neighbors (between 1 and the number of training samples in each CV fold). Up to 16 search iterations per parameter.

Logistic Regression (LogReg) Parameter space searched over: regularization of the error term (log-uniform distribution between 10^{-9} and 10^{3}). Up to 16 search iterations per parameter.

Performance Metric

The F1-Score is used to precisely measure how well each classifier is able to correctly classify samples. CNM use multiple metrics and calibrate them to the same scale for comparison, and although this is comprehensive, it is not done here in lieu of parsimony. F1-Score was chosen primarily because of its use in CNM (which allows the comparison of results here with their results) and, despite its biases and flaws, it is considered convention in machine learning classification problems (Sokolova, Japkowicz, and Szpakowicz 2006). Should the measure have been included in CNM, it would have been more suitable to use Cohen's Kappa or a variation thereof (Ben-David 2008).

Data Set	Samples	Features
ADULT	30162	14/104
CELL	6235	50
COD	20000	15/60
MUSH	8124	22/117

Table 1: Summary of the Data Sets

Data Sets

The classifiers above were assessed over four data sets:

- ADULT and MUSH are from the UCI Machine Learning Repository (Dheeru and Karra Taniskidou 2017), and ADULT was assessed in CNM. The goal of ADULT is to classify whether a person has an income of over $50,000, given demographic and career information. The goal of MUSH is to classify whether a certain mushroom is edible or not, given appearance information.
- CELL is a data set of images of blood cells, labeled by one of four types of cell, from Kaggle.com. Because there are four classes, this is made a binary classification problem by retaining only Eosinophil and Monocyte cell images. Given an image of a small set of blood cells, the goal is to correctly classify which type of blood cell is stained. Each image is converted to 32-bit floating point pixels (gray scale) and flattened into vector form. Because this would result in 76,000 features, the data are reduced to 50 features with Principal Component Analysis (PCA). PCA is regarded as a standard dimension reduction algorithm for images. 50 features were chosen because more dimensions did not explain significantly more of the remaining variance in the data.
- COD is a data set taken from (Craven and Shavlik 1993) and the UCI Machine Learning Repository, and represents the binary classification (coding or noncoding) of DNA sequences of 15 nucleotides. COD was also assessed in CNM's work.

Rows in ADULT and MUSH that contained null values were removed. Any ordinal or nominal features in the data sets were one-hot encoded for Lin-SVM, RBF-SVM, KNN, and LogReg. Moreover, because not all features in ADULT were one-hot encoded, they lie on vastly different scales. Thus, each feature is appropriately scaled to mean 0 and variance 1. This prevents algorithms such as KNN from being heavily influenced by only a small portion of the feature set. Rows that contained null or unknown values were removed. Table 1 presents quantitative details about each data set. The Features column includes the original feature count and the one-hot encoded feature count.

Bayesian Optimization

Optimizing the hyper-parameters for each classifier is traditionally done with an exhaustive grid search. Not only is this computationally expensive, but it is naive and does not fully take advantage of the true underlying parameter space. (Bergstra et al. 2011) introduced a variety of different hyper-parameter optimization methods including a Gaussian Process approach, a Tree-structured Parzen Estimator approach, Random Search, and Sequential Search. Later, (Bergstra and Bengio 2012) proved

that a random search over a hyper-parameter space was definitively more efficient than a grid search. They observed that when searching for neural network hyper-parameters, this technique could find as good as or better models than grid search in only a fraction of the time. (Snoek, Larochelle, and Adams 2012) advanced the idea of Bayesian hyper-parameter tuning, offering advice and proving its competitiveness with expert human hyper-parameter tuning.

With this in mind, the `skopt` module of the `scikit-optimize` Python package was utilized. The parameter search is optimized with Gaussian Processes and the number of iterations is specified for each classifier (and enumerated above). Moreover, with Bayesian optimization, it is possible to specify an early stopping routine. While this dramatically improves the efficiency of the overall search, it is easy to stop too soon. For this reason, the mean change over the most recent seven iterations must be less than 10^{-5} for the algorithm to move on. The settings for this hyper-parameter search and the iteration and early stopping requirements were modeled on the examples provided by the creators of the `skopt` module, however the Discussion section will assess this initialization further.

In mathematical notation, this would equate to having a black box classifier $f(x)$, an acquisition function $u(x)$, and satisfying:

$$x_{t+1} = \text{argmin}_x u(x)$$

where $t \in 1, ..., T$ and T being the number of iterations total. The acquisition function for the purposes here is the Expected Improvement function, where $u(x) = -\text{EI} = \mathbb{E}[f(x) - f(x'_t)]$ and $f(x'_t)$ is the best observation thus far. In theory, this maximizes the information the next sample should provide and improve algorithm convergence performance.

Classification Performance

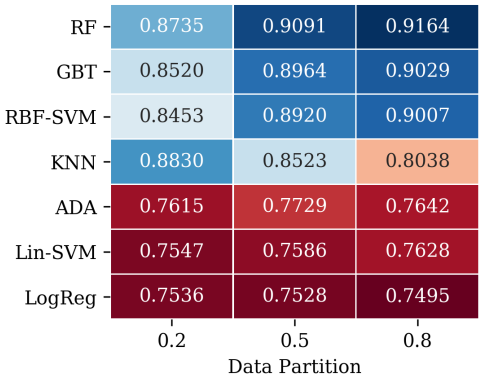
Overall, the classification algorithms used performed very well on the data sets. The F1-Score results achieved here are similar to those of CNM. Random Forest, Gradient Boosted Trees, and RBF-SVM all lie at the top. Whereas KNN and Logistic Regression are in the middle and lower ground, respectively. As shown by Fig 1, the better performing classifiers improved with more training. The weaker classifiers performed as well with 20% of the data set held out for training as with 80%. The performance of KNN decreased with more data.

The training time necessary for the classifiers evaluated widely varied. Some, like LogReg and Lin-SVM, were able to execute in seconds on even the largest data sets, whereas others, like RBF-SVM and KNN, took between 1-8 hours, depending on the training set size. Unfortunately, the classifiers were not evaluated over equal computation allocations, thus duration would be difficult to compare. Moreover, this prevents us from quantitatively assessing performance improvements as a result of utilizing Bayesian hyper-parameter search.

The hyper-parameters which resulted in the highest testing F1-Score for the 0.5 partition over each data set are reported in the Appendix. Floating point values are rounded to four decimals in consideration of space. The full results may be found in the logs contained in the code repository.[1] Because only ranges are specified for the hyper-parameter spaces, many of these results appear arbitrary and uneven. This is the nature of the optimization method and is comparable to a Random Search.

[1]Code at https://github.com/liebscher/cogs118a-model-comparison

Figure 1: Average F1-Score for all data sets across classifiers



Discussion

At first glance, the performance results in Table 2 shed light on a few interesting conclusions. The results closely reflect those of CNM, which further validates beliefs about each classifier.

To begin, the MUSH data set, across all partitions, had an overall testing F1-Score very near 1.0 for all classifiers, which says a remarkable amount about the data themselves. After exploring past results though, it does not seem so unusual to have such high scores (Ali and Smith 2006). Image data do not seem to be handled well by several classifiers, including AdaBoost, Linear SVM, and Logistic Regression. The latter two may have performed so poorly in comparison with the other algorithms because they are linear classifiers. This severely limits their ability to understand the data, especially since the data are being represented by their PCA transformation and this likely places them in a nontrivial space. However, the results of AdaBoost are harder to interpret. The performance of AdaBoost was relatively poor across the other data sets, so either the hyper-parameters were initialized or optimized poorly, or the general performance of the algorithm is poor. Three hyper-parameters are critical for this algorithm to function: the base estimator, the total number of estimators, and the learning rate. The base estimator for this boosting algorithm was a decision tree of maximum depth 1. The latter two hyper-parameters were optimized in cross validation, thus it may have made sense to modify the underlying estimator to elicit improved performance. However, this changes the classifier drastically and would have needed to be treated as a separate classifier all-together.

The performance results also suggest there was some overfitting on the ADULT data set. RBF-SVM, Random Forest, and KNN all train with very high F1-Scores but very low validation scores. They then evaluate decently well on the test set, comparable with the other classifiers. This may imply that during training, these classifiers learn the known structure very well but have poor generalization power. This is likely a result of the parameters of the classifiers. For RBF-SVM, there is a regularization parameter to control the influence of non-separable classes. Looking at the

Model	Part	MUSH			ADULT			COD			CELL		
		Train	Val	Test	Train	Val	Test	Train	Val	Test	Train	Val	Test
ADA	0.2	0.9880	0.9873	0.9853	0.7289	0.6856	0.8579	0.7060	0.6987	0.6907	0.9037	0.5753	0.5121
	0.5	1.000	0.9999	1.000	0.7079	0.6962	0.8624	0.7117	0.7087	0.7142	0.8575	0.5713	0.5151
	0.8	1.000	1.000	1.000	0.6631	0.6600	0.8584	0.7022	0.6990	0.6989	0.6502	0.5528	0.4996
GBT	0.2	1.000	1.000	1.000	0.8687	0.6957	0.8601	0.8979	0.7532	0.7577	1.000	0.7877	0.7902
	0.5	1.000	1.000	1.000	0.7954	0.7053	0.8669	0.8311	0.7713	0.7818	1.000	0.9241	0.9370
	0.8	1.00	1.000	1.000	0.7697	0.7118	0.8725	0.7794	0.7794	0.7792	1.000	0.9437	0.9599
Lin-SVM	0.2	1.000	0.9996	0.9994	0.6717	0.6587	0.8323	0.7236	0.7167	0.6963	0.6237	0.5059	0.4908
	0.5	1.000	1.000	1.000	0.6717	0.6675	0.8221	0.7099	0.7050	0.7157	0.5386	0.5166	0.4967
	0.8	1.000	1.000	1.000	0.6695	0.6652	0.8428	0.7081	0.7056	0.7100	0.5211	0.5190	0.4983
RBF-SVM	0.2	1.000	0.9996	0.9996	0.9502	0.5125	0.7911	0.9994	0.7623	0.7590	1.000	0.8026	0.8314
	0.5	1.000	1.000	1.000	0.9389	0.6267	0.8349	0.9267	0.7544	0.7345	1.000	0.9905	0.9984
	0.8	1.000	1.000	1.000	0.8483	0.6596	0.8457	0.8696	0.7623	0.7569	1.000	0.9998	1.000
RF	0.2	1.000	1.000	0.9996	1.000	0.6711	0.8450	1.000	0.7448	0.7491	1.000	0.8652	0.9004
	0.5	1.000	1.000	1.000	0.9999	0.6802	0.8528	0.772	0.7724	0.7897	1.000	0.9900	0.9939
	0.8	1.000	1.000	1.000	0.9999	0.6823	0.8552	1.000	0.7936	0.8116	1.000	0.9978	0.9989
KNN	0.2	1.000	0.9998	1.000	1.000	0.5756	0.8100	1.000	0.7010	0.7283	1.000	0.9860	0.9935
	0.5	1.000	1.000	1.000	0.9999	0.4536	0.8166	1.000	0.7121	0.7483	1.000	0.8534	0.8444
	0.8	1.000	1.000	1.000	0.9999	0.3714	0.7915	1.000	0.7171	0.7585	1.000	0.6788	0.6651
LogReg	0.2	1.000	0.9998	0.9993	0.6806	0.6666	0.8290	0.7255	0.7159	0.7034	0.5890	0.5005	0.4825
	0.5	1.000	1.000	1.000	0.6775	0.6718	0.8371	0.7118	0.7047	0.7139	0.5402	0.4715	0.4602
	0.8	1.000	1.000	1.000	0.6700	0.6660	0.8414	0.7088	0.7068	0.7094	0.5156	0.4428	0.4472

Table 2: Average F1-Score over three trials among all classifiers, data sets, and data partitions

parameter choices of KNN says that it often chooses $k = 1$ or $k = |\text{kfold training set}|$. It would be worth it to perform more iterations of each classifier to overcome these local minima in the Bayesian Search.

In retrospect, a few improvements could have been made. Most importantly it seems, the early stopping criteria could have been made more strict. There was some variance (± 0.15) in the testing scores after cross-validation for the three classifiers mentioned above. KNN performance across data partitions may have decreased because its complex training cycles may have prevented it from getting past the early stopping criteria.

In the future, and for later studies which compare multiple classifiers, it would be wise to compute an ANOVA test between the classifiers to test for statistically significant improvements between them (Demšar 2006). It would have also been wise to calculate and calibrate multiple test metrics. However, these would require far more computational power and time than was permitted for this exploration.

Conclusion

Overall, the performance of seven classification algorithms was evaluated across four data sets. For thoroughness, each classifier was tested over three independent trials, where each trial was subject to three partitions of the data. For each partition of the data, 5-fold cross validation was performed on the training and validation sets. For each fold, an optimal set of hyper-parameters was found for the classifier using Bayesian Search. Contrary to traditional grid search, this method improves performance, takes advantage of the underlying parameter space with specific priors, and reduces redundant and insignificant searches. Performance overall proved that Random Forests, Gradient Boosted Trees, and RBF-SVM achieve high results. K-Nearest Neighbors may also be a viable solution but should be treated with care and precision. Unsurprisingly, linear classifiers perform well only if the data are linearly separable. Future work should address overfitting and lack of convergence on the global minimum by allowing more time for training. Additionally, a more rigorous comparison should be performed to truly assess the ranking of the classifiers.

Acknowledgments

The author is especially grateful to the teaching staff of Cognitive Science 118A of Fall 2018, most notably Professor Zhuowen Tu. Additionally, I thank my friend Samuel Blake for his valuable feedback on ideas. Without the data provided free of charge and easily accessible from the UCI Machine Learning Repository, this project would not have come to fruition.

References

Ahmed, N. K.; Atiya, A. F.; Gayar, N. E.; and El-Shishiny, H. 2010. An empirical comparison of machine learning models for time series forecasting. *Econometric Reviews* 29(5-6):594–621.

Ali, S., and Smith, K. A. 2006. On learning algorithm selection for classification. *Applied Soft Computing* 6(2):119–138.

Ben-David, A. 2008. Comparison of classification accuracy using cohens weighted kappa. *Expert Systems with Applications* 34(2):825–832.

Bergstra, J., and Bengio, Y. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research* 13(Feb):281–305.

Bergstra, J. S.; Bardenet, R.; Bengio, Y.; and Kégl, B. 2011. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, 2546–2554.

Caruana, R., and Niculescu-Mizil, A. 2006. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on Machine learning*, 161–168. ACM.

Chen, T., and Guestrin, C. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 785–794. ACM.

Craven, M. W., and Shavlik, J. W. 1993. Learning to represent codons: A challenge problem for constructive induction. In *IJCAI*, 1319–1324. Citeseer.

Demšar, J. 2006. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research* 7(Jan):1–30.

Dheeru, D., and Karra Taniskidou, E. 2017. UCI machine learning repository.

King, R. D.; Feng, C.; and Sutherland, A. 1995. Statlog: comparison of classification algorithms on large real-world problems. *Applied Artificial Intelligence an International Journal* 9(3):289–333.

Kotsiantis, S. B.; Zaharakis, I.; and Pintelas, P. 2007. Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering* 160:3–24.

Lemke, C.; Budka, M.; and Gabrys, B. 2015. Metalearning: a survey of trends and technologies. *Artificial intelligence review* 44(1):117–130.

Snoek, J.; Larochelle, H.; and Adams, R. P. 2012. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, 2951–2959.

Sokolova, M.; Japkowicz, N.; and Szpakowicz, S. 2006. Beyond accuracy, f-score and roc: a family of discriminant measures for performance evaluation. In *Australasian joint conference on artificial intelligence*, 1015–1021. Springer.

	MUSH	ADULT	COD	CELL
ADA	learning_rate: 0.2874 n_estimators: 740	learning_rate: 0.8013 n_estimators: 608	learning_rate: 0.7756 n_estimators: 791	learning_rate: 1.4214 n_estimators: 906
GBT	n_estimators: 470	n_estimators: 876	n_estimators: 881	n_estimators: 1024
Lin-SVM	C: 1000.0	C: 1.6396e-07	C: 0.8062	C: 14.5320
RBF-SVM	C: 20.9248 gamma: 0.0235	C: 7.3954e-05 gamma: 0.1537	C: 1.5924 gamma: 0.34618	C: 537.8010 gamma: 0.1004
RF	max_features: 18	max_features: 4	max_features: 2	max_features: 36
KNN	n_neighbors: 1 weights: uniform	n_neighbors: 1451 weights: distance	n_neighbors: 1461 weights: distance	n_neighbors: 1 weights: uniform
LogReg	C: 21.6154	C: 0.00015	C: 11.4002	C: 2.3602e-06

Table 3: Hyper-parameters for the estimators with the best test scores on the 0.5 partition