The Potential Benefits of Data Set Filtering and Learning Algorithm Hyperparameter Optimization

Michael R. Smith, Tony Martinez, and Christophe Giraud-Carrier

Department of Computer Science, Brigham Young University, Provo, UT 84602 USA
msmith@axon.cs.byu.edu, martinez@cs.byu.edu, cgc@cs.byu.edu
http://axon.cs.byu.edu

Abstract. The quality of a model induced by a learning algorithm is dependent upon the training data *and* the hyperparameters supplied to the learning algorithm. Prior work has shown that a model's quality can be significantly improved by filtering out low quality instances or by tuning the learning algorithm hyperparameters. The potential impact of filtering and hyperparameter optimization (HPO) is largely unknown. In this paper, we estimate the *potential* benefits of instance filtering and HPO. While both HPO and filtering significantly improve the quality of the induced model, we find that filtering has a greater potential effect on the quality of the induced model than HPO, motivating future work in filtering.

1 Introduction

Given a set of training instances composed of input feature vectors and corresponding labels, the goal of supervised machine learning is to induce an accurate generalizing function (hypothesis) that maps feature vectors to labels. The quality of the induced function is dependent on the learning algorithm's hyperparameters *and* the quality of the training data. It is known that no learning algorithm or hyperparameter setting is best for all data sets (no free lunch theorem [26]) and that the performance of many learning algorithms is sensitive to their hyperparameter settings. It is also well-known that real-world data sets are typically noisy.

Prior work has shown that the generalization performance of an induced model can be significantly improved through hyperparameter optimization (HPO) [1], or by increasing the quality of the training data using techniques such as noise correction [11], instance weighting [17], or instance filtering [20]. Searching the hyperparameter space and improving the quality of the training data have generally been examined in isolation and the potential impact of their usage has not been examined. In this paper, we compare the effects of HPO with the effects of improving the quality of the training data through filtering. The results of our experiments provide insight into the potential effectiveness of both HPO and filtering.

We evaluate 6 commonly used learning algorithms and 46 data sets. We examine the effects of HPO and filtering by: 1) using a standard approach that selects the hyperparameters of an algorithm by maximizing the accuracy on a validation set and 2) using an optimistic approach that sets the hyperparameters for an algorithm using the 10-fold cross-validation accuracy. The standard and optimistic approaches are explained in

more detail in Section 4. Essentially, the optimistic approach indicates how well a technique *could* perform if the training set were representative of the test set and provides insight into the *potential* benefit of a given technique. The standard approach provides a representative view of HPO and filtering in their present state and allows an evaluation of how well current HPO and filtering techniques fulfill their potential.

Using the standard approach, we find that in most cases both HPO and filtering significantly increase classification accuracy over using a learning algorithm with its default parameters trained on unfiltered data. For the optimistic estimates of HPO and filtering, we find that *filtering significantly improves the classification accuracy over HPO* for all of the investigated learning algorithms–increasing the accuracy more than HPO for almost all of the considered data sets. HPO achieves an average accuracy of 84.8% while filtering achieves an average accuracy of 89.1%. The standard approach for HPO and filtering achieves an average accuracy of 82.6% and 82.0% respectively. These results provide motivation for further research into developing algorithms that improve the quality of the training data.

2 Related Work

Smith et al. [21] found that a significant number of instances are difficult to classify correctly, that the hardness of each instance is dependent on its relationship with the other instances in the training set and that some instances can be detrimental. Thus, there is a need for improving how detrimental instances are handled during training as they affect the classification of other instances. Improving the quality of the training data has typically fallen into three approaches: filtering, cleaning, and instance weighting [7].

Each technique within an approach differs in how detrimental instances are identified. A common technique for filtering removes instances from a data set that are misclassified by a learning algorithm or an ensemble of learning algorithms [3]. Removing the training instances that are suspected to be noise and/or outliers prior to training has the advantage that they do not influence the induced model and generally increase classification accuracy. A negative side-effect of filtering is that beneficial instances can also be discarded and produce a worse model than if all of the training data had been used [18]. Rather than discarding the instances from a training set, noisy or possibly corrupted instances can be cleaned or corrected [11]. However, this could artificially corrupt valid instances. Alternatively, weighting weights suspected detrimental instances rather than discards them and allows for an instance to be considered on a continuum of detrimentality rather than making a binary decision [17].

Other methods exist for improving the quality of the training data, such as feature selection/extraction [8]. While feature selection and extraction can improve the quality of the training data, we focus on improving quality via filtering – facilitating a comparison between filtering and HPO on the same feature set.

Much of the previous work in improving the quality of the training data artificially corrupts training instances to determine how well an approach would work in the presence of noisy or mislabeled instances. In some cases, a given approach *only* has a significant impact when there are large degrees of artificial noise. In contrast, we do not artificially corrupt a data set to create detrimental instances. Rather, we seek to identify

the detrimental instances that are already contained in a data set and show that correctly labeled, non-noisy instances can *also* be detrimental for inducing a model of the data. Properly handling detrimental instances can result in significant gains in accuracy.

The grid search and manual search are the most common types of HPO techniques in machine learning and a combination of the two approaches are commonly used [12]. Bergstra and Bengio [1] proposed to use a random search of the hyperparameter space. The premise of random HPO is that most machine learning algorithms have very few hyperparameters that considerably affect the final model while the other hyperparameters have little to no effect. Random search provides a greater variety of the hyperparameters that considerably affect the model. Given the same amount of time constraints, random HPO has been shown to outperform a grid search. Random search, while providing improvements over a grid-search, is unreliable for tuning the hyperparameters for some learning algorithms such as deep belief networks [2]. Bayesian optimization has also been used to search the hyperparameter space [23]. Bayesian optimization techniques model the dependence of an error function $\mathcal{E}$ on the hyperparameters λ as $p(\mathcal{E}|\lambda)$ using, for example, random forests [10] or Gaussian processes [23].

3 Preliminaries

Let T represent a training set composed of a set of input vectors $X = \{x_1, x_2, \ldots, x_n\}$ and corresponding label vectors $Y = \{y_1, y_2, \ldots, y_n\}$, i.e., $T = \{\langle x_i, y_i\rangle : x_i \in X \wedge y_i \in Y\}$. Given that in most cases, all that is known about a task is contained in the set of training instances T, at least initially, the training instances are generally considered equally. Most machine learning algorithms seek to induce a hypothesis $h : X \rightarrow Y$ that minimizes a specified loss function $\mathcal{L}(\cdot)$. As most real-world data sets contain some level of noise, there is generally a model-dependent regularization term $\mathcal{R}(\cdot)$ added to $\mathcal{L}(\cdot)$ that penalizes more complex models and aids in overfit avoidance. The noise in T may arise from errors in the data collection process such as typos or errors in data collection equipment. In addition to noise from errors, there may be non-noisy outlier instances due to the stochastic nature of the task. A hypothesis h is induced by a learning algorithm g trained on T with hyperparameters λ ($h = g(T, \lambda)$), such that:

$$h^* = \underset{h \in \mathcal{H}}{\operatorname{argmin}} \frac{1}{|T|} \sum_{\langle x_i, y_i\rangle \in T} \mathcal{L}(h(x_i), y_i) + \alpha \mathcal{R}(h) \tag{1}$$

where α is a regularization parameter greater than or equal to 0 that determines how much weight to apply to the regularization term and $h(\cdot)$ returns the predicted class for a given input. The quality of the induced hypothesis h is characterized by its empirical error for a specified error function $\mathcal{E}$ on a test set V: $E(h, V) = \frac{1}{|V|} \sum_{\langle x_i, y_i\rangle \in V} \mathcal{E}(h(x_i), y_i)$ where V can be T or a disjoint set of instances. In k-fold cross-validation, the empirical error is the average empirical error from the k folds (i.e., $1/k\ E(h_i, V_i)$).

Characterizing the success of a learning algorithm at the data set level (e.g., accuracy or precision) optimizes over the entire training set and marginalizes the impact of a single training instance on an induced model. Some sets of instances can be more beneficial than others for inducing a model of the data and some can even be detrimental. By *detrimental instances*, we mean instances that have a negative impact on

the induced model. For example, outliers or mislabeled instances are not as beneficial as border instances and are detrimental in many cases. In addition, other instances can be detrimental for inducing a model of the data even if they are labeled correctly. Formally, a set $\mathcal{D}$ of detrimental instances is a subset of the training data that, when used in training, increases the empirical error, i.e., $E(g(T, \lambda), V) > E(g(T - \mathcal{D}, \lambda), V)$.

The effect of training with detrimental instances is demonstrated in the hypothetical two-dimensional data set shown in Figure 1. Instances A and B represent detrimental instances. The solid line represents the "actual" classification boundary and the dashed line represents a potential induced classification boundary. Instances A and B adversely affect the induced classification boundary because they "pull" the classification boundary and cause several other instances to be misclassified that otherwise would have been classified correctly.

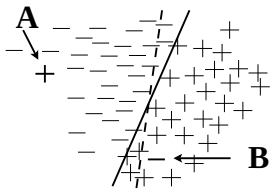

Fig. 1. Hypothetical 2-dimensional data set that shows the potential effects of detrimental instances in the training data on a learning algorithm.

Despite most learning algorithms having a mechanism to avoid overfitting, the presence of detrimental instances may still affect the induced model for many learning algorithms. Mathematically, the effect of each instance on the induced hypothesis is shown in Equation 1. The loss from each instance in T, including detrimental instances, is equally weighted. Detrimental instances have the most significant impact during the early stages of training where it is difficult to identify them [6]. The presence of $\mathcal{D}$ may also affect the value of $\mathcal{R}(h)$. For example, removing $\mathcal{D}$ from T could produce a "simpler" h that reduces $\mathcal{R}(h)$.

3.1 Hyperparameter Optimization

The quality of an induced model by a learning algorithm depends in part on the learning algorithm's hyperparameters. With hyperparameter optimization (HPO), the hyperparameter space Λ is searched to minimize the empirical error on V:

$$\underset{\lambda \in \Lambda}{\operatorname{argmin}} E(g(T, \lambda), V). \tag{2}$$

The hyperparameters can have a significant effect on the quality of the induced model as well as suppressing the effects of detrimental instances. For example, in a support vector machine, [4] use the ramp-loss function which limits the penalty on instances that are too far from the decision boundary rather than the more typical 0-1 loss function to

handle detrimental instances. Suppressing the effects of detrimental instances with HPO improves the induced model, but does not change the fact that detrimental instances *still* affect the model. Each instance is still considered during the learning process though its influence may be lessened. We describe the method we use for HPO in Section 4.1.

3.2 Filtering

The quality of an induced model also depends on the quality of the training data where, for example, the quality of the training data can be measured by the amount of detrimental instances present. Low quality training data results in lower quality induced models. Improving the quality of the training data involves searching the training set space to find an optimal subset that minimizes the empirical error:

$$\underset{t \in \mathcal{P}(T)}{\operatorname{argmin}} E(g(t, \lambda), V)$$

where t is a subset of T and $\mathcal{P}(T)$ is the power set of T. The removed instances obviously have no effect on the induced model. In Section 4.2, we describe how we identify detrimental instances and search for an optimal subset of the training data that minimizes empirical error.

4 Implementation Details

4.1 Bayesian Hyperparameter Optimization

In this paper, we use Bayesian optimization for HPO. Specifically, we use *sequential model-based optimization* (SMBO) [10] as it has been shown to yield better performance than grid and random search [23,24]. SMBO is a stochastic optimization framework that builds a probabilistic model $\mathcal{M}$ that captures the dependence of $\mathcal{E}$ on λ. SMBO first initializes $\mathcal{M}$. After initializing $\mathcal{M}$, SMBO searches the search space by 1) querying $\mathcal{M}$ for a promising λ to evaluate, 2) evaluating the loss $\mathcal{E}$ of using configuration λ, and then 3) updating $\mathcal{M}$ with λ and $\mathcal{E}$. Once the budgeted time is exhausted, the hyperparameter configuration with the minimal loss is returned.

To select a candidate hyperparameter configuration, SMBO relies on an acquisition function $a_{\mathcal{M}} : \Lambda \rightarrow \mathbb{R}$ which uses the predictive distribution of $\mathcal{M}$ to quantify how useful knowledge about λ would be. SMBO maximizes $a_{\mathcal{M}}$ over Λ to select the most useful hyperparameter configuration λ to evaluate next. One of the most prominent acquisition functions is the *positive expected improvement* (EI) over an existing error rate $\mathcal{E}_{min}$ [19]. If $\mathcal{E}(\lambda)$ represents the error rate of hyperparameter configuration λ, then the EI function over $\mathcal{E}_{min}$ is: $EI_{\mathcal{E}_{min}}(\lambda) = max\{\mathcal{E}_{min} - \mathcal{E}(\lambda), 0\}$. As $\mathcal{E}(\lambda)$ is unknown, the expectation of $\mathcal{E}(\lambda)$ with respect to the current model $\mathcal{M}$ can be computed as: $\mathbb{E}_{\mathcal{M}}[EI_{\mathcal{E}_{min}}(\lambda)] = \int_{-\infty}^{\mathcal{E}_{min}} max\{\mathcal{E}_{min} - \mathcal{E}, 0\} \cdot p(\mathcal{E}|\lambda) d\mathcal{E}$.

SMBO is dependent on the model class used for $\mathcal{M}$. Following [24], we use sequential model-based algorithm configuration (SMAC) [10] for $\mathcal{M}$ with EI as $a_{\mathcal{M}}$, although others could be used such as the tree-structured Parzen estimator. To model $p(\mathcal{E}|\lambda)$, we use random forests as they tend to perform well with discrete and continuous input data.

Using random forests, SMAC obtains a predictive mean μ_λ and variance σ_λ^2 of $p(\mathcal{E}|\lambda)$ calculated using the predictions from the individual trees in the forest for λ. $p(\mathcal{E}|\lambda)$ is then modeled as a Gaussian distribution $\mathcal{N}(\mu_\lambda, \sigma_\lambda^2)$. To create diversity in the evaluated configurations, every second configuration is selected at random as suggested [24]. For k-fold cross-validation, the **standard approach** finds the hyperparameters that minimize the error for each of the k validation sets as shown in Equation 2. The **optimistic approach** finds the hyperparameters that minimize the *k*-fold cross-validation error: $\text{argmin}_{\lambda \in \Lambda} \frac{1}{k} E(g(T_i, \lambda), V_i)$ where T_i and V_i are the training and validation sets for the ith fold. The hyperparameter space Λ is searched using Bayesian hyperparameter optimization for both approaches.

4.2 Filtering

Identifying detrimental instances is a non-trivial task. Fully searching the space of subsets of training instances generates 2^N subsets of training instances where N is the number of training instances. Even for small data sets, it is computationally infeasible to induce 2^N models to determine which instances are detrimental. There is no known way to determine how a set of instances will affect the induced classification function from a learning algorithm without inducing a classification function with the investigated set of instances removed from the training set.

The Standard Filtering Approach ($\mathcal{G}$-Filter) Previous work in noise handling has shown that class noise (e.g. mislabeled instances) is more detrimental than attribute noise [15]. Thus, searching for detrimental instances that are likely to be misclassified is a natural place to start. In other words, we search for instances where the probability of the class label is low given the feature values (i.e., low $p(y_i|x_i)$). In general, $p(y_i|x_i)$ does not make sense outside the context of an induced hypothesis. Thus, using an induced hypothesis h from a learning algorithm trained on T, the quantity $p(y_i|x_i)$ can be approximated as $p(y_i|x_i, h)$. After training a learning algorithm on T, the class distribution for an instance x_i can be estimated based on the output from the learning algorithm. Prior work has examined removing instances that are misclassified by a learning algorithm or an ensemble of learning algorithms [3]. We filter instances using an ensemble filter that removes instances that are misclassified by more than $x\%$ of the algorithms in the ensemble.

The dependence of $p(y_i|x_i, h)$ on a particular h can be lessened by summing over the space of all possible hypotheses:

$$p(y_i|x_i) = \sum_{h \in \mathcal{H}} p(y_i|x_i, h) p(h|T). \tag{3}$$

However, this formulation is infeasible to compute in most practical applications as $p(h|T)$ is generally unknown and $\mathcal{H}$ is large and possibly infinite. To sum over $\mathcal{H}$, one would have to sum over the complete set of hypotheses, or, since $h = g(T, \lambda)$, over the complete set of learning algorithms and their associated hyperparameters.

The quantity $p(y_i|x_i)$ can be estimated by restricting attention to a diverse set of representative algorithms (and hyperparameters). The diversity of the learning algorithms refers to the likelihood that the learning algorithms classify instances differently.

Table 1. Set of learning algorithms $\mathcal{G}$ used to estimate $p(y_i|x_i)$.

LEARNING ALGORITHMS
* MULTILAYER PERCEPTRON TRAINED WITH BACK PROPAGATION (MLP)
* DECISION TREE (C4.5)
* LOCALLY WEIGHTED LEARNING (LWL)
* 5-NEAREST NEIGHBORS (5-NN)
* NEAREST NEIGHBOR WITH GENERALIZATION (NNGE)
* NAÏVE BAYES (NB)
* RIPPLE DOWN RULE LEARNER (RIDOR)
* RANDOM FOREST (RANDFOREST)
* REPEATED INCREMENTAL PRUNING TO PRODUCE ERROR REDUCTION (RIPPER)

A natural way to approximate the unknown distribution $p(h|T)$ is to weight a set of representative learning algorithms, and their associated hyperparameters, $\mathcal{G}$, a priori with an equal, non-zero probability while treating all other learning algorithms as having zero probability. We select a diverse set of learning algorithms using unsupervised metalearning (UML) [13] to get a good representation of $\mathcal{H}$, and hence a reasonable estimate of $p(y_i|x_i)$. UML uses Classifier Output Difference (COD) [16] measures the diversity between learning algorithms as the probability that the learning algorithms make different predictions. UML clusters the learning algorithms based on their COD scores with hierarchical agglomerative clustering. Here, we consider 20 commonly used learning algorithms with their default hyperparameters as set in Weka [9]. A cut-point of 0.18 was chosen to create nine clusters and a representative algorithm from each cluster was used to create $\mathcal{G}$ as shown in Table 1.

Given a set $\mathcal{G}$ of learning algorithms, we approximate Equation 3 to the following:

$$p(y_i|x_i) \approx \frac{1}{|\mathcal{G}|} \sum_{j=1}^{|\mathcal{G}|} p(y_i|x_i, g_j(T, \lambda)) \tag{4}$$

where $p(h|T)$ is approximated as $\frac{1}{|\mathcal{G}|}$ and g_j is the jth learning algorithm from $\mathcal{G}$. As not all learning algorithms produce probabilistic outputs, the distribution $p(y_i|x_i, g_j(T, \lambda))$ is estimated using the Kronecker delta function in this paper.

The Optimistic Filtering Approach ($\mathcal{A}$-Filter) To measure the *potential* impact of filtering, we need to know how removing an instance or set of instances affects the generalization capabilities of the model. We measure this by dynamically creating an ensemble filter from $\mathcal{G}$ using a greedy algorithm for a given data set and learning algorithm. This allows us to find a specific ensemble filter that is best for filtering a given data set and learning algorithm combination. The adaptive ensemble filter is constructed by iteratively adding the learning algorithm g from $\mathcal{G}$ that produces the highest cross-validation classification accuracy when g is added to the ensemble filter. Because we are using the probability that an instance will be misclassified rather than a binary yes/no decision (Equation 4), we also use a threshold ϕ to determine which instances are detrimental. Instances with a $p(y_i|x_i)$ less than ϕ are discarded from the training set. A

constant threshold value for ϕ is set to filter the instances for all iterations. The baseline accuracy for the adaptive approach is the accuracy of the learning algorithm without filtering. The search stops once adding one of the remaining learning algorithms to the ensemble filter does not increase accuracy, or all of the learning algorithms in $\mathcal{G}$ have been used.

Even though all of the detrimental instances are included for evaluation, the adaptive filter ($\mathcal{A}$-Filter) overfits the data since the cross-validation accuracy is used to determine which set of learning algorithms to use in the ensemble filter. This allows us to find the detrimental instances to examine the effects that they can have on an induced model. This is not feasible in practical settings, but provides insight into the potential improvement gained from filtering.

5 Filtering and HPO

In this section, we compare the effects of filtering with those of HPO using the optimistic and standard approaches presented in Section 4. The optimistic approach provides an approximation of the potential of HPO and filtering. In addition to reporting the average classification accuracy, we also report the average rank of each approach. The average accuracy and rank for each algorithm is determined using 5 by 10-fold cross-validation. Statistical significance between pairs of algorithms is determined using the Wilcoxon signed-ranks test (as suggested by [5]) with an alpha value of 0.05.

5.1 Experimental Methodology

For HPO, we use the version of SMAC implemented in auto-WEKA [24] as described in Section 4.1 Auto-WEKA searches the hyperparameter spaces for the learning algorithms in the Weka machine learning toolkit [9] for a specified amount of time. To estimate the amount of time required for a learning algorithm to induce a model of the data, we ran our selected learning algorithms with ten random hyperparameter settings and calculated the average and max running times. On average, a model was induced in less than 3 minutes. The longest time required to induce a model was 845 minutes. Based on this analysis, we run auto-WEKA for one hour for most of the data sets. An hour long search explores more than 512 hyperparameter configurations for most of the learning algorithm/data set combinations. The time limit is adjusted accordingly for the larger data sets. Following [24], we run four runs with different random seeds provided to SMAC.

For filtering using the ensemble filter ($\mathcal{G}$-filter), we use thresholds ϕ of 0.5, 0.7, and 0.9. Instances that are misclassified by more than $\phi\%$ of the learning algorithms are removed from the training set. The $\mathcal{G}$-filter uses all of the learning algorithms in the set $\mathcal{G}$ (Table 1). The accuracy on the test set from the value of ϕ that produces the highest accuracy on the training set is reported.

To show the effect of filtering detrimental instances and HPO on an induced model, we examine filtering and HPO in six commonly used learning algorithms (MLP trained with backpropagation, C4.5, kNN, Naïve Bayes, Random Forest, and RIPPER) on a set of 46 UCI data sets [14]. The LWL, NNge, and Ridor learning algorithms are not used

Table 2. Results for maximizing the 10-fold cross-validation accuracy for HPO and filtering.

	MLP	C4.5	*k*NN	NB	RF	RIP
ORIG	82.28 (2.98)	81.30 (2.91)	80.56 (2.74)	77.66 (2.70)	82.98 (2.89)	79.86 (2.96)
HPO	86.37 (1.87)	84.25 (1.96)	83.89 (2.22)	80.89 (1.96)	86.81 (1.85)	82.08 (1.80)
VS ORIG	45,0,1	42,1,3	34,1,11	34,0,12	44,0,2	**46,0,0**
A-FILTER	**89.96 (1.13)**	**88.74 (1.09)**	**91.14 (1.02)**	**82.74 (1.30)**	**91.02 (1.20)**	**88.16 (1.24)**
VS ORIG	**46,0,0**	**46,0,0**	**46,0,0**	**44,2,0**	**43,3,0**	44,0,2
VS HPO	39,1,6	41,1,4	45,0,1	32,0,14	37,0,9	37,0,9

for analysis because they do not scale well with the larger data sets–not finishing due to memory overflow or large amounts of running time.[1]

5.2 Optimistic Approach

The optimistic approach indicates how well a model *could* generalize on novel data. Maximizing the cross-validation accuracy is a type of overfitting. However, using 10-fold cross-validation accuracy for HPO and filtering, essentially measures the generalization capability of a learning algorithm for a given data set.

The results comparing the potential benefits of HPO and filtering are shown in Table 2. Each section gives the average accuracy and average rank for each learning algorithm as well as the number of times the algorithm is greater than, equal to, or less than a compared algorithm. HPO and the adaptive filter significantly increase the classification accuracy for all of the investigated learning algorithms. The values in bold represent if HPO or the adaptive filter is significantly greater than the other. For all of the investigated learning algorithms, the $\mathcal{A}$-filter significantly increases the accuracy over HPO. The closest the two techniques come to each other is for NB, where the $\mathcal{A}$-filter achieves an accuracy of 82.74% and an average rank of 1.30 while HPO achieves an accuracy of 80.89% and an average rank of 1.96. For all learning algorithms other than NB, the average accuracy is about 89% for filtering and 84% for HPO. Thus, filtering has a greater potential for increase in generalization accuracy. The difficulty lies in how to find the optimal set of training instances.

As might be expected, there is no set of learning algorithms that is the optimal ensemble filter for all algorithms and/or data sets. Table 3 shows the frequency for which a learning algorithm with default hyperparameters was selected for filtering by the $\mathcal{A}$-filter. The greatest percentage of cases an algorithm is selected for filtering for each learning algorithm is in bold. The column "ALL" refers to the average from all of the learning algorithms as the base learner. No instances are filtered in 5.36% of the cases. Thus, given the right filter, filtering to some extent increases the classification accuracy in about 95% of the cases. Furthermore, random forest, NNge, MLP, and C4.5 are the most commonly chosen algorithms for inclusion in the ensemble filter. However, no one learning algorithm is selected in more than 27% of the cases. The filtering algorithm that is most appropriate is dependent on the data set and the learning algorithm.

[1] For the data sets on which the learning algorithms did finish, the effects of HPO and filtering on LWL, NNge, and Ridor are consistent with the other learning algorithms.

Table 3. The frequency of selecting a learning algorithm when adaptively constructing an ensemble filter. Each row gives the percentage of cases that an algorithm was included in the ensemble filter for the learning algorithm in the column.

	ALL	MLP	C4.5	kNN	NB	RF	RIP
NONE	5.36	2.69	2.95	3.08	5.64	5.77	1.60
MLP	18.33	16.67	15.77	20.00	**25.26**	23.72	16.36
C4.5	17.17	17.82	15.26	22.82	14.49	13.33	20.74
5NN	12.59	11.92	14.23	1.28	10.00	17.18	16.89
LWL	6.12	3.59	3.85	4.36	23.72	3.33	3.59
NB	7.84	5.77	6.54	8.08	5.13	10.26	4.92
NNGE	19.49	**26.67**	21.15	21.03	11.15	**24.74**	23.40
RF	**21.14**	22.95	**26.54**	**23.33**	15.77	15.13	**24.20**
RID	14.69	14.87	16.79	18.33	11.92	16.54	12.77
RIP	8.89	7.82	7.69	8.85	13.08	7.44	4.39

This coincides with the findings from [18] that the efficacy of noise filtering in the nearest-neighbor classifier is dependent on the characteristics of the data set. Understanding the efficacy of filtering and determining *which* filtering approach to use for a given algorithm/data set is a direction of future work.

Analysis. In some cases HPO achieves a lower accuracy than orig, showing the complexity of HPO. The $\mathcal{A}$-Filter, on the other hand, never fails to improve the accuracy. Thus, higher quality data can compensate for hyperparameter settings and suggests that the instance space may be less complex and/or richer than the hyperparameter space. Of course, filtering does not outperform HPO in all cases, but it does so in the majority of cases.

5.3 Standard Approach

The previous results show the potential impact of filtering and HPO. We now examine HPO and filtering using the standard approach to highlight the need for improvement in filtering. The results comparing the $\mathcal{G}$-filter, HPO, and using the default hyperparameters trained on the original data set are shown in Table 4. HPO significantly increases the classification accuracy over not using HPO for all of the learning algorithms. Filtering significantly increases the accuracy for all of the investigated algorithms except for random forests. Comparing HPO and the G-Filter, only HPO for naïve Bayes and random forests significantly outperforms the $\mathcal{G}$-filter.

Analysis. In their current state, HPO and filtering generally improve the quality of the induced model. The results justify the computational overhead required to run HPO. Despite these results, using the default hyperparameters result in higher classification accuracy for 11 of the 46 data sets for C4.5, 12 for kNN, and 12 for NB, highlighting the complexity of searching over the hyperparameter space Λ. The effectiveness of HPO is dependent on the data set as well as the learning algorithm. Typically, as was done here, a single filtering technique is used for a set of data sets with no model of the dependence of a learning algorithm on the training instances. The accuracies for

Table 4. Performance comparison of using the default hyperparameters, HPO, and the $\mathcal{G}$-filter.

	MLP	C4.5	*k*NN	NB	RF	RIP
ORIG	82.28 (2.54)	81.3 (2.13)	80.56 (2.26)	77.66 (2.28)	82.98 (2.28)	79.86 (2.46)
HPO	83.08 (1.78)	82.42 (1.76)	82.85 (1.59)	**81.11 (1.63)**	**84.72 (1.54)**	81.15 (1.74)
VS ORIG	32,0,14	29,1,16	31,5,10	31,2,13	34,0,12	30,2,14
G-FILTER	84.17 (1.61)	81.9 (1.96)	81.61 (2.00)	79.49 (2.02)	83.49 (2.17)	81.25 (1.72)
VS ORIG	39,0,7	23,5,18	27,1,18	28,1,17	25,0,21	37,0,9
VS HPO	22,3,21	19,1,26	17,1,28	16,0,30	13,0,33	20,2,24

filtering and HPO are significantly lower than the optimistic estimate given in Section 5.2 motivating future work in HPO and especially in filtering.

6 Conclusion

In this paper, we compared the potential benefits of filtering with HPO. HPO may reduce the effects of detrimental instances on an induced model but the detrimental instances are still considered in the learning process. Filtering, on the other hand, removes the detrimental instances–completely eliminating their effects on the induced model.

We used an optimistic approach to estimate the potential accuracy of each method. Using the optimistic approach, both filtering and HPO significantly increase the classification accuracy for all of the considered learning algorithms. However, *filtering has a greater potential effect* on average, increasing the classification accuracy from 80.8% to 89.1% on the observed data sets. HPO increases the average classification accuracy to 84.8%. Future work includes developing models to understand the dependence of the performance of learning algorithms given the instances used for training. To better understand how instances affect each other, we are examining the results from machine learning experiments stored in repositories that include which instances were used for training and their predicted class [25,22]. We hope that the presented results provide motivation for improving the quality of the training data.

References

1. J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.
2. J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2546–2554. Curran Associates, Inc., 2011.
3. C. E. Brodley and M. A. Friedl. Identifying mislabeled training data. *Journal of Artificial Intelligence Research*, 11:131–167, 1999.
4. R. Collobert, F. Sinz, J. Weston, and L. Bottou. Trading convexity for scalability. In *Proceedings of the 23rd International Conference on Machine learning*, pages 201–208, 2006.
5. J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

6. J. L. Elman. Learning and development in neural networks: The importance of starting small. *Cognition*, 48:71–99, 1993.
7. B. Frénay and M. Verleysen. Classification in the presence of label noise: a survey. *IEEE Transactions on Neural Networks and Learning Systems*, 25(5):845–869, 2014.
8. I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.
9. M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.
10. F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the International Learning and Intelligent Optimization Conference*, pages 507–523, 2011.
11. J. Kubica and A. Moore. Probabilistic noise identification and data cleaning. In *Proceedings of the 3rd IEEE International Conference on Data Mining*, pages 131–138, 2003.
12. H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th International Conference on Machine Learning*, pages 473–480, 2007.
13. J. Lee and C. Giraud-Carrier. A metric for unsupervised metalearning. *Intelligent Data Analysis*, 15(6):827–841, 2011.
14. M. Lichman. UCI machine learning repository, 2013.
15. D. F. Nettleton, A. Orriols-Puig, and A. Fornells. A study of the effect of different types of noise on the precision of supervised learning techniques. *Artificial Intelligence Review*, 33(4):275–306, 2010.
16. A. H. Peterson and T. R. Martinez. Estimating the potential for combining learning models. In *Proceedings of the ICML Workshop on Meta-Learning*, pages 68–75, 2005.
17. U. Rebbapragada and C. E. Brodley. Class noise mitigation through instance weighting. In *Proceedings of the 18th European Conference on Machine Learning*, pages 708–715, 2007.
18. J. A. Sáez, J. Luengo, and F. Herrera. Predicting noise filtering efficacy with data complexity measures for nearest neighbor classification. *Pattern Recognition*, 46(1):355–364, 2013.
19. M. Schonlau, W. J. Welch, and D. R. Jones. *Global versus local search in constrained optimization of computer models*, volume Volume 34 of *Lecture Notes–Monograph Series*, pages 11–25. Institute of Mathematical Statistics, Hayward, CA, 1998.
20. M. R. Smith and T. Martinez. Improving classification accuracy by identifying and removing instances that should be misclassified. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, pages 2690–2697, 2011.
21. M. R. Smith, T. Martinez, and C. Giraud-Carrier. An instance level analysis of data complexity. *Machine Learning*, 95(2):225–256, 2014.
22. M. R. Smith, A. White, C. Giraud-Carrier, and T. Martinez. An easy to use repository for comparing and improving machine learning algorithm usage. In *Proceedings of the 2014 International Workshop on Meta-learning and Algorithm Selection (MetaSel)*, pages 41–48, 2014.
23. J. Snoek, H. Larochelle, and R. Adams. Practical bayesian optimization of machine learning algorithms. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2951–2959. 2012.
24. C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-weka: combined selection and hyperparameter optimization of classification algorithms. In *proceedings of the 19th International Conference on Knowledge Discovery and Data Mining*, pages 847–855, 2013.
25. J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo. Openml: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013.
26. D. H. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8(7):1341–1390, 1996.