Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2019) 27: 1534 – 1545
© TÜBİTAK
doi:10.3906/elk-1807-212

Optimal training and test sets design for machine learning

Burkay GENÇ[*], **Hüseyin TUNÇ**
Department of Policy and Strategy Studies, Hacettepe University, Ankara, Turkey

Received: 19.07.2018 • **Accepted/Published Online:** 11.02.2019 • **Final Version:** 22.03.2019

Abstract: In this paper, we describe histogram matching, a metric for measuring the distance of two datasets with exactly the same features, and embed it into a mixed integer programming formulation to partition a dataset into fixed size training and test subsets. The partition is done such that the pairwise distances between the dataset and the subsets are minimized with respect to histogram matching. We then conduct a numerical study using a well-known machine learning dataset. We demonstrate that the training set constructed with our approach provides feature distributions almost the same as the whole dataset, whereas training sets constructed via random sampling end up with significant differences. We also show that our method introduces neither positive nor negative bias in prediction accuracy of a decision tree—used as a representative example of a machine learning method.

Key words: Distribution matching, instance selection, training set selection, optimization

1. Introduction

History of machine learning dates back to 1950s, if we do not classify earlier statistical advances as machine learning studies. In his seminal paper, the computer science pioneer Alan Turing, for the first time, mentioned "learning machine" in 1950 [1]. However, it was not until 2010s that machine learning was recognized as a practical solution approach to many real-life problems. This was mostly due to the high hardware requirements of machine learning algorithms which have been satisfied recently.

Roughly speaking, machine learning algorithms aim to extract information from data on hand and produce viable prediction models. The general framework of any machine learning study can be summarized as follows. A machine learning study starts with a dataset that contains existing data about the problem to be studied. This data is then split into three parts: training set, validation set and test set. Next, a machine learning model is somehow constructed from the training set. This is the "learning" part where the machine learning algorithm analyzes the training set to establish a mathematical model representing the set itself. Next, this model is tested on the validation set to evaluate its performance. At this stage, if the resulting performance levels are not acceptable, the learning step is revisited to change the algorithm parameters to revise the model. Once the validation performance reaches to a prescribed acceptable level, the model is evaluated on the test set. The results obtained here represent the final score of the model and cannot be further tweaked.

Training, validation, and test sets are often selected based on simple random sampling with prescribed ratios. Usually, these ratios are determined as 70/15/15, where 70% of the data goes to training, 15% goes to validation, and the remaining 15% goes to testing. Other ratios may also be used based on the volume and

[*]Correspondence: burkay.genc@hacettepe.edu.tr

variety of the data. Usually, it is preferable to keep as much data as possible for the training set to obtain a stronger model.

Most machine learning algorithms, if not all, rely heavily on the selected training, validation, and test sets. In particular, the results produced by the underlying tool could dramatically be different as these sets change from one case to another. This is particularly evident for certain tools such as decision trees and boosting. The performance of a robust algorithm can even be negatively affected due to an unfortunate splitting. This obviously raises an important question regarding the reliability of the results produced by the underlying machine learning algorithm within a framework where training, validation, and test sets are randomly drawn. In other words, we cannot be sure about how good a machine learning algorithm results will be if we obtain different training, validation, and test sets every time we start from scratch.

To clarify things further, let us consider the following toy dataset in Table 1. Here, there are 8 people (observations) each of which has 3 distinct features, i.e. age, gender, and salary. For simplicity, assume that we aim to split the entire set into two subsets: training and test sets.

Table 1. An example dataset.

Obs.	Age	Gender	Salary
1	20	M	High
2	20	M	Low
3	20	F	Low
4	20	F	Low
5	20	M	High
6	40	F	High
7	40	F	High
8	40	F	Low

A first look at this data shows that, on average, older people have high salaries while younger people have low salaries, and males have high salaries while females have low salaries. Now, assume that we have randomly drawn 6 observations to build a training set as in Table 2. If we consider this training set, we must now conclude that neither age nor gender has a significant effect on salary. Moreover, we may also argue that an average teenager earns a high salary, and the same holds for the average female. Hence, it seems like a female teenager is very likely to have a high salary. Considering the remaining observations in rows 3 and 4 as a test set, we face a major problem: both are young females with low salaries. As a result, our predictions failed for all test cases yielding an impressive 100% error rate.

Table 2. A randomly sampled training set.

Obs.	Age	Gender	Salary
1	20	M	High
2	20	M	Low
5	20	M	High
6	40	F	High
7	40	F	High
8	40	F	Low

Although this example contains exaggerations to a certain extent, it can successfully reflect the very core of the problem: the evaluation scores obtained in a machine learning study depends on how we split the training and test sets. It is possible to increase or decrease the final error rates simply by splitting the sets in a different way as is the case in random sampling.

One may even think that by randomly producing many training sets, it is possible to train a model with a lower error rate. However, this approach leads us to an overwhelmingly biased model performing great with the test set but fails significantly with any other given test set. This issue is known as "overfitting" and must be avoided to increase the accuracy of the model.

By now, it should be clear that, this anomaly is due to the fact that training and test sets, and the whole dataset are all characteristically different. Unfortunately, it is not trivial to formally define the term "characteristically different". In the example above, the difference lies in the fact that the ratios of the levels of the target feature within the sampled subset with respect to the input features does not match (or represent) the ratios within the original dataset. In another example, the culprit may be the difference of variances, correlations, or some other statistical measures between the original dataset and the subsets. Having an unbalanced class distribution for the target variable, also known as "underrepresentation", is yet another reason for producing a weak model with a high error variance. All the above mentioned issues then result in a robustness problem for the resulting model.

In this paper, we address the problem of designing the training and test sets aiming to eliminate the aforementioned robustness problem once and for all. To the best of our knowledge, this problem has not been considered in the literature earlier. However, there are studies on how to reduce the size of a given dataset. The literature reflects a clear dichotomy on how to reduce the size of a dataset between feature selection and instance selection. The former refers to reducing the number of features in the dataset (i.e. columns in a tabular view), whereas the latter stands for reducing the number of observations (i.e. instances or rows in a tabular view). Feature selection aims to find an input dataset by reducing the irrelevant and/or low impact features (variables) without diminishing prediction efficiency. To this end, a variety of approaches has been proposed including filter methods [e.g., 2–5] and wrapper methods [e.g., 6–9]. Here, we refer the interested reader to [10] for a detailed survey on feature selection methods. Instance selection, on the other hand, is often referred to decreasing the number of observations (rows) in a given training set. This is mostly needed due to the space and time complexities inherited in machine learning tools. Here, the idea is to find a condensed training set out of a given larger training set so as to increase the efficiency of the approach without sacrificing the predictive power of the underlying method. To this end, the literature provides a variety of approaches. There are instance selection methods deliberately designed for specific classifiers including $K-$nearest neighbors [e.g., 11–15] and support vector machine [e.g., 16–18]. Also, there are studies on more general methods mostly constructed on a prescribed distance metrics and applications [e.g., 19–23]. Here, the idea is to somehow cluster the instances based on their pairwise distances so as to determine the critical instances in terms of prediction. Nevertheless, none of these approaches indeed take care of the aforementioned robustness issue occurred due to the selection of the training and test sets.

Here, we introduce an approach constructed on matching the distributions of datasets. In particular, we propose a heuristic measure to evaluate the difference between the datasets and show how one can employ optimization method to split a dataset (population) into fixed-size training and test subsets so as to minimize the difference between the distribution of the population and those of the subsets. We provide a mixed integer programming formulation of the problem to solve the optimization problem. In our numerical study, we illustrate the dispersion of the error induced due to random sampling of the training set. We also demonstrate how our approach perform on an actual machine learning scenario. More specifically, by means of a simulation study, we show that a decision tree trained by our subsets results in neither positive nor negative error bias. Our results reflect that the resulting error of our approach is indeed found to be located around the median of the entire error space of the simulation study conducted.

The remainder of the paper is organized as follows. In Section 2, we provide the problem description, set the notation, and provide the mixed integer programming formulation. In Section 3, we introduce our numerical study and discuss our findings. Finally, we provide conclusive remarks in Section 4.

2. Data and algorithms

Data comes in many formats; nonetheless, it is often stored and presented in tabular form. It is presented by means of rows and columns in tabular format in which each row of data refers to a single observation, whereas each column corresponds to a feature. Each feature has a specific format, usually one of numeric, character, categorical, or logical. Numeric features may take any value within a range of numbers in a continuous fashion. Categorical features may only have a fixed number of distinct and predetermined values. Logical features are similar to categorical ones; however they are limited to take only two different values, such as “true” and “false”. Finally, character features cover what is not covered by the other types. In this paper, we consider datasets where each feature is categorical.

2.1. Methodology

Suppose that we have a dataset (i.e. population) with N observations and a set of features $\mathbb{C}$ each of which has a set of levels $\mathsf{K_j}$, where $j \in \mathbb{C}$. In this study, we consider a problem on how to split the entire dataset into two mutually exclusive fixed size subsets forming a partition—a training set H having h observations (i.e. $|H| = h$) and a test set having $N - h$ observations. To do so, we provide a novel approach and develop its computational method to select unique training and test sets establishing a natural benchmark for the given dataset. The proposed approach is simply built on the idea of matching the distributions of training and test sets to the distribution of the population itself. Eventually, we hereby argue that the training and test sets which respectively possess very similar probability distributions with that of the population would naturally provide a convenient frame for a machine learning algorithm. To this end, we convert this problem to an optimization problem and develop its mixed integer programming formulation (MIP) which will be introduced later on.

An important question to consider here is how to measure the similarity between two distributions, in our case the population and the selected training and/or test sets. The conventional methods such as Kolmogorov–Smirnov and chi-square tests would not fit for serving our aim since they mostly require a specific family of distributions or even univariate distributions. Also, even if it is applicable, it is not easy to implement those conventional tests in a multivariate setting. Furthermore, it is also not trivial to embed those tests into a mathematical model. Therefore, in this study, we adopt a simple yet pragmatic approach to measure the similarity, i.e. hereby referred to as “histogram matching”. This approach simply assumes each variate (i.e. feature) is independent and treat them as if they are indeed univariate distributions. Then, the aim is to measure the distance, $D^{Y,Z}$, between two distributions Y and Z by the following,

$$D^{Y,Z} = \sum_{j \in \mathbb{C}} \sum_{t \in \mathsf{K_j}} \left| P_j^Y(X = t) - P_j^Z(X = t) \right|, \tag{1}$$

where $P_j^Y(X=t)$ represents the probability of having the value of t for feature j in distribution Y, whereas $P_j^Z(X=t)$ stands for that of Z. This distance metric can obviously be regarded as a proxy since it ignores the relation within features.

As stated earlier, we want to select the optimal training and test sets by simply minimizing the total distance between their distributions and that of population. Here, the following results enable us to simplify our analysis:

Proposition 1 *Given a dataset S with n observations, and a partition of S into S_1 and S_2 with n_1 and n_2 observations, respectively; the following holds for any level $t \in \mathsf{K_j}$ and any feature $j \in \mathbb{C}$:*

$$P_j^{\mathsf{S}}(X=t)n = P_j^{\mathsf{S}_1}(X=t)n_1 + P_j^{\mathsf{S}_2}(X=t)n_2. \tag{2}$$

Proof The left hand side of Eq. (2) provides the frequency of observations in S, where the observed level of feature j has the value of t. Similarly, the first and second terms on the right hand side respectively give the frequency of t values in S_1 and S_2. Since S_1 and S_2 defines a partition on S of S, the frequency on the left is equal to the sum of the frequencies on the right. □

We can now provide the following derivation:

$$P_j^{\mathsf{S}_2}(X=t)n_2 = P_j^{\mathsf{S}}(X=t)n - P_j^{\mathsf{S}_1}(X=t)n_1$$
$$P_j^{\mathsf{S}_2}(X=t) = \frac{P_j^{\mathsf{S}}(X=t)n - P_j^{\mathsf{S}_1}(X=t)n_1}{n-n_1}$$

Using Eq. (1),

$$\begin{aligned}
D^{\mathsf{S},\mathsf{S}_2} &= \sum_{j\in\mathbb{C}}\sum_{t\in\mathsf{K_j}} \left|P_j^{\mathsf{S}}(X=t) - P_j^{\mathsf{S}_2}(X=t)\right| \\
&= \sum_{j\in\mathbb{C}}\sum_{t\in\mathsf{K_j}} \left|P_j^{\mathsf{S}}(X=t) - \frac{P_j^{\mathsf{S}}(X=t)n - P_j^{\mathsf{S}_1}(X=t)n_1}{n-n_1}\right| \\
&= \sum_{j\in\mathbb{C}}\sum_{t\in\mathsf{K_j}} \left|\frac{-P_j^{\mathsf{S}}(X=t)n_1 + P_j^{\mathsf{S}_1}(X=t)n_1}{n-n_1}\right| \\
&= \frac{n_1}{n-n_1}\sum_{j\in\mathbb{C}}\sum_{t\in\mathsf{K_j}} \left|P_j^{\mathsf{S}}(X=t) - P_j^{\mathsf{S}_1}(X=t)\right|.
\end{aligned}$$

Hence,

$$D^{\mathsf{S},\mathsf{S}_2} = \frac{n_1}{n_2}D^{\mathsf{S},\mathsf{S}1}. \tag{3}$$

The following lemma is based on this result:

Lemma 1 *Given a dataset S with n observations, and a subset $\mathsf{S}_1 \in \mathsf{S}$ of size n_1; S_1 is closest to S among all subsets of size n_1 with respect to D, if and only if $\mathsf{S}-\mathsf{S}_1$ is closest to S.*

Proof Eq. (3) clearly shows that for a fixed n_1/n_2 ratio, the distance between S and $\mathsf{S} - \mathsf{S}_1$ is linearly dependent on the distance between S and S_1. Hence, minimizing one is indeed equivalent to minimizing the other. □

This lemma explicitly shows that once the optimal training set minimizing the total distance between its distribution and that of the population is found, the remaining set used for the test will be optimal in the same sense. As such, this observation enables us to restrict our attention solely on selection of the training set.

To be able to compute the distance, we need to compute the probability mass function of each feature in a given dataset within a mathematical model. To this end, we employ the following binary parameter defined for each $i \in \{1, \ldots, N\}$, $j \in \mathbb{C}$, and $t \in \mathsf{K_j}$;

$$x_{jti} = \begin{cases} 1 & \text{if } a_{ij} = t \\ 0 & \text{otherwise,} \end{cases}$$

where a_{ij} stands for the entry of observation i and feature j in the dataset.

Here, x_{jti} simply takes the value of 1 if an observation i has the value of t in its feature j. The probability mass function of each feature can now be easily expressed as follows;

$$P_j(X = t) = \frac{\sum_{i=1}^{n} x_{jti}}{n}, \tag{4}$$

where n is the number of observations in the set considered.

Now, let P_j^Y denote the probability mass function of the entire dataset (i.e. the population), whereas P_j^Z denotes that of a subset (i.e. the training set) in Eq. (1). Then, we can easily compute the total distance between the training set and the population and rewrite Eq. (1) via Eq. (4) as follows:

$$\sum_{j \in \mathbb{C}} \sum_{t \in \mathsf{K_j}} \left| \frac{\sum_{i=1}^{N} x_{jti}}{N} - \frac{\sum_{i \in H} x_{jti}}{h} \right|. \tag{5}$$

Having established these, we now focus on the mixed integer programming formulation of concern. The problem entails selecting the observations fixed size in number for the training set so as to minimize the total distance between training set distribution and the population. We construct our model by using the following decision variables:

A_{jt}	absolute value of the difference between the probability of level $t \in \mathsf{K_j}$ of feature $j \in \mathbb{C}$ in the population and that of training set,
y_i	binary variable that equals 1 if observation i is chosen in the training set, and 0 otherwise.

Notice that the definition given above explicitly states $A_{jt} = |P_j^Y(X = t) - P_j^Z(X = t)|$. Let us, for now, omit the fact that the absolute value is not a linear function. Later on, we introduce a set of constraints to represent absolute value function with linear expressions. Also, recall that the objective of the problem is to minimize the total distance between the population and the training set. Then, we can write the objective function as follows:

$$\min \quad \sum_{j \in \mathbb{C}} \sum_{t \in \mathsf{K_j}} A_{jt}. \tag{6}$$

Next, we can introduce the set of constraints ensuring that the decision variable A_{jt} indeed equal to absolute error between two distributions. To do so, we need to compute the probability values of the population and the training set for each feature and level. The former is constant, but the latter, in fact, depends on our training set selection. Therefore, we can compute the probability values of the training set by simply conditioning each observation with decision variable y_i. Also, to be able to make sure that A_{jt} takes the absolute value of the gap, we bound A_{jt} from below by simply negative and positive values of the gap itself. These can be written as follows:

$$h\sum_{i=1}^{N} x_{jti} - N\sum_{i=1}^{N} x_{jti}y_i \le NhA_{jt} \quad \forall j \in \mathbb{C}, t \in \mathsf{K_j}, \tag{7}$$

$$N\sum_{i=1}^{N} x_{jti}y_i - h\sum_{i=1}^{N} x_{jti} \le NhA_{jt} \quad \forall j \in \mathbb{C}, t \in \mathsf{K_j}\,. \tag{8}$$

As mentioned earlier, we want to select a training set having h observations. This can be expressed as

$$\sum_{i=1}^{N} y_i = h. \tag{9}$$

This finalizes our MIP model. For convenience, we provide the entire model below:

$$\begin{aligned} \min \quad & Eq.~(6) \\ \text{subject to} \quad & Eqs.~(7)-(9) \\ & A_{jt} \ge 0 \quad \forall j \in \mathbb{C}, t \in \mathsf{K_j} \\ & y_i \in \{0,1\} \quad \forall i \in \{1,\ldots,N\} \end{aligned}$$

3. Results and evaluation

The aim of this numerical study was to demonstrate the efficiency of the proposed methods compared to using simple random sampling in generating training and test tests. In particular, we provided an illustrative example and provide some test statistics to evaluate the performance of our method. The implementation was done in two parts. The optimization part was carried out on a 3.40 GHz Intel Core i7-3770 CPU with 16 GB RAM. Gurobi v6.5 was used as an MIP solver. We terminated the solver when the objective value reached to 1e-4, i.e. the so-called objective value to stop optimization. The data analysis part, on the other hand, was conducted using the R language and various R add-on packages.

For testing purposes, we used the *mushroom* dataset obtained from UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/mushroom). The dataset contains 8124 mushroom instances observed in 22 categorical features. The target feature codes whether the mushroom is edible or not. The remaining features represent attributes related to size, color, shape etc. To demonstrate the effectiveness of our method, we produced training sets with different sizes varying between 500 and 6500, with incremental steps of size 500. For each set size, we generated one optimized training set by using our method—abbreviated as OS, and randomly drawn 500 samples (randomly generated training sets). We then compared the histogram matching errors between the generated training sets and the entire dataset computed via Eq. (1). Table 3 outlines the results in tabular form. In this table, we provide the summary statistics of histogram matching errors for

Table 3. Distribution matching errors obtained by random sampling and optimized training set.

	Random sampling error						
Size	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	OS Err.
500	0.714	0.9819	1.1014	1.1235	1.2601	1.7903	0.0572
1000	0.4845	0.6774	0.7433	0.7651	0.8423	1.3166	0.0301
1500	0.3611	0.536	0.592	0.6072	0.6727	1.0569	0.0211
2000	0.3122	0.4412	0.4908	0.5018	0.5544	0.8516	0.0139
2500	0.2817	0.3827	0.4274	0.4333	0.4759	0.7327	0.0107
3000	0.2301	0.3314	0.3718	0.3787	0.4176	0.6467	0.0118
3500	0.197	0.2926	0.3244	0.3296	0.3613	0.5193	0.0117
4000	0.1749	0.2582	0.2858	0.2939	0.3201	0.4973	0.0089
4500	0.1633	0.2249	0.2542	0.2582	0.286	0.3962	0.0089
5000	0.1348	0.1982	0.2215	0.2261	0.2458	0.3567	0.0068
5500	0.1166	0.1751	0.1944	0.1994	0.2198	0.3295	0.008
6000	0.105	0.1482	0.1663	0.1702	0.1865	0.2907	0.0073
6500	0.0949	0.1241	0.1386	0.1423	0.1557	0.23	0.0094

randomly drawn 500 samples and present the error of our method for each training size. We can clearly see from this table that our method produces training sets with much smaller distribution matching errors. For example, with a training set size of 2500 observations, the minimum error obtained by random sampling is 26 times larger than that of OS. If we consider the median error, the rate becomes 40. Even with very large training set sizes, such as 6500 (corresponding to 80% of the original dataset), OS is matching the whole dataset 10 times better than the best set produced by random sampling.

We further provide some plots to visualize the distribution of histogram matching errors obtained by the random sampling and that of OS in Figure 1. In this figure, the density curves represent the error distribution and the dashed vertical line represents the error of OS for 4 selected training set sizes. In all four plots, it is easy to observe OS establishes a natural benchmark and provides far better training sets compared to random sampling.

We can also focus on a specific training set size and a feature to see how OS provides a much better distribution match compared to random sampling. Here, we randomly draw a training set—abbreviated as RS, and compare it with the corresponding OS. In Figures 2–5, we demonstrate examples of level frequencies for different training set sizes and features. In each figure, we look at a specific feature and a training set size. The x axis represents the different categoric levels of that feature, and the y axis represents the percentage error of OS and RS for each level of the feature. The percentage error represents the magnitude of the difference between the training set's level frequency and the population's level frequency as compared to the population's level frequency. Suppose that a level has a frequency of 0.10 within the population. An error percentage of 50% then means that the frequency of the level within the training set is either 0.05 or 0.15.

For example, in Figure 2, we consider the gill-color feature for a training set size of 500. The gill-color feature contains 12 different levels, represented by single character labels on the x axis. There are two bars for each level: the left represents OS, whereas the right refers to RS. We can observe that, for almost all levels, OS matches the population significantly better than RS does. Moreover, RS can produce errors as large as 100%, which could introduce a significant bias in any machine learning algorithm.

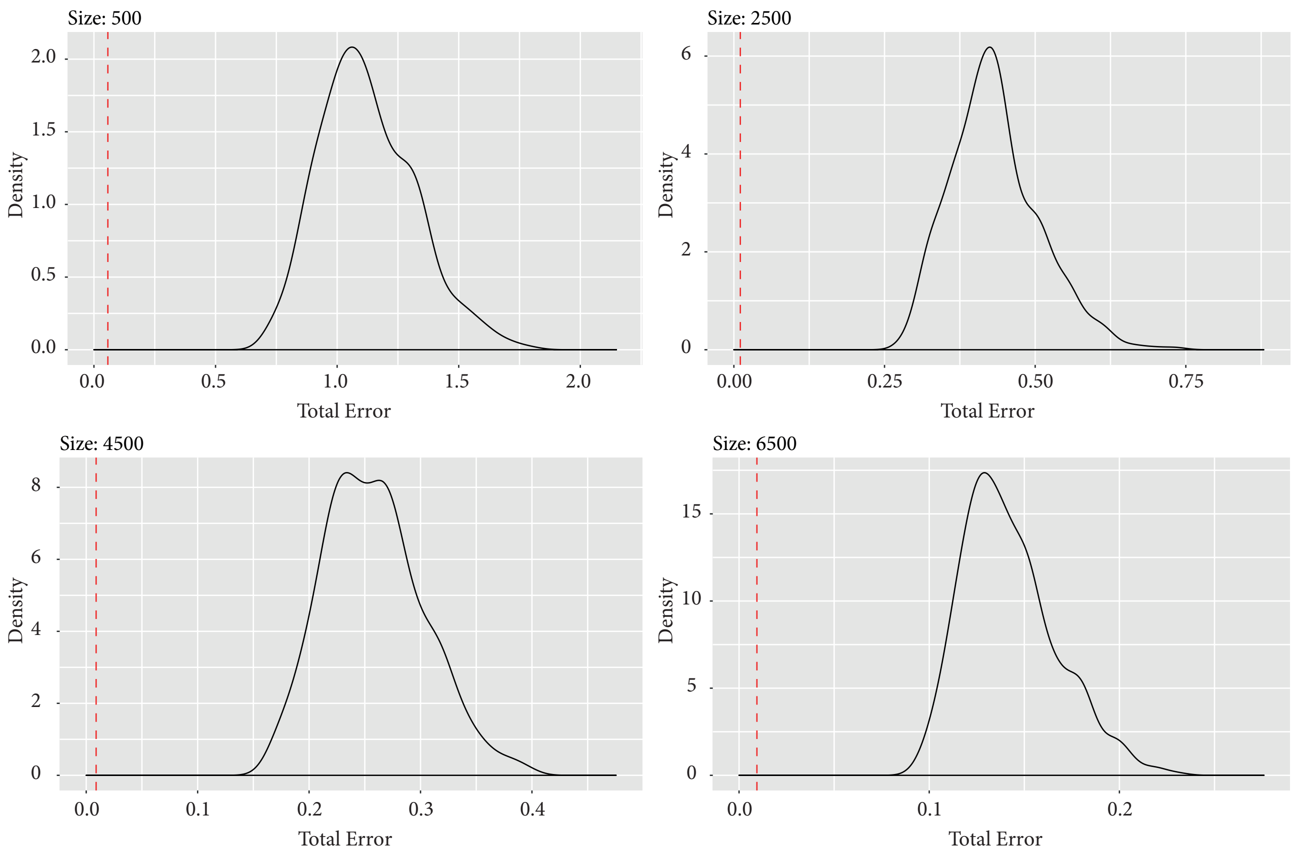

Figure 1. Comparison of error distributions of RS and OS for selected sample sizes.

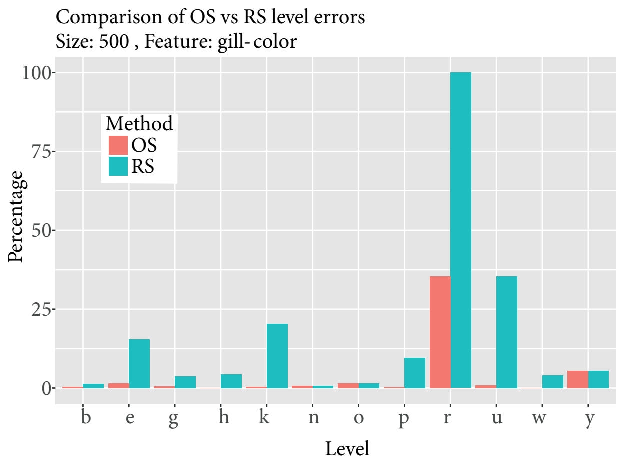

Figure 2. Comparison of OS and RS level errors for training set size 500 and gill-color feature.

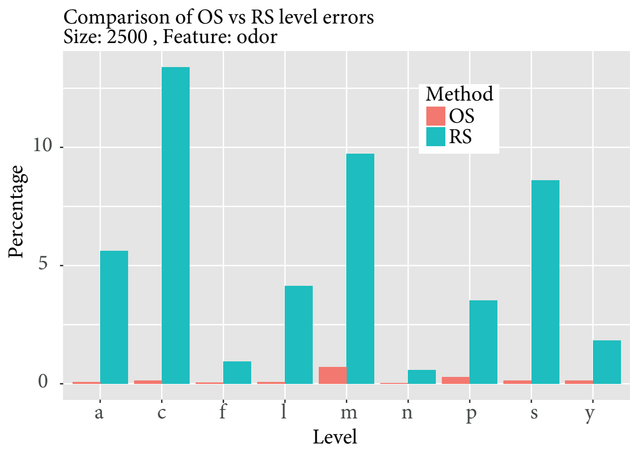

Figure 3. Comparison of OS and RS level errors for training set size 2500 and odor feature.

In Figure 3, we look at odor levels for a training set size of 2500. Again, OS is significantly better than RS on all levels. In Figures 4 and 5, we consider features of cap-color and edible for sample sizes of 4500 and 6500, respectively. The results show a similar pattern as in the previous features.

Finally, we turned our attention to what OS could introduce to an actual machine learning algorithm.

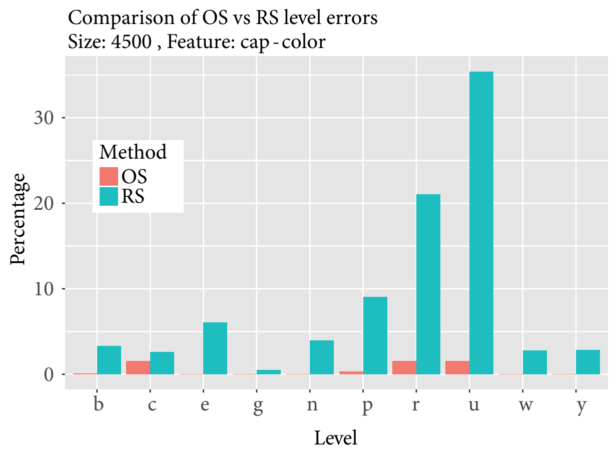

Figure 4. Comparison of OS and RS level errors for training set size 4500 and cap-color feature.

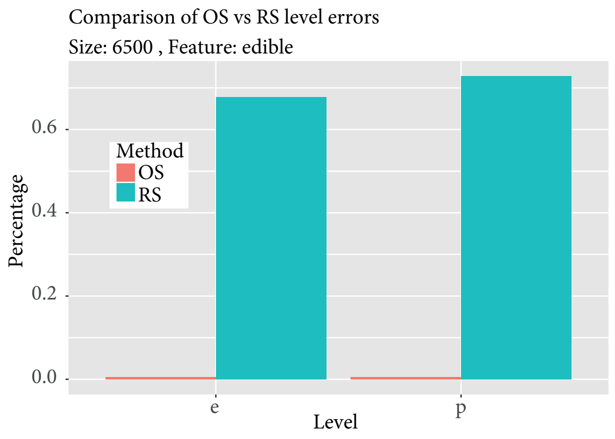

Figure 5. Comparison of OS and RS level errors for training set size 6500 and edibility feature.

To this end, we used four different training sizes: 500, 2500, 4500, and 6500. For each training set size considered, we randomly generated 5000 training sets. Then, we used the CART (classification and regression trees) algorithm to construct a decision tree using these samples and OS. We plotted classification error of OS (represented by the dashed lines in the figure) and the random samples on the same plot in Figure 6. In this figure, we provide 4 boxplots side by side with varying training set sizes. We can now see how the decision tree classification accuracy changes as the training set size increases. We can also observe the placement of OS' classification error among that of random samples.

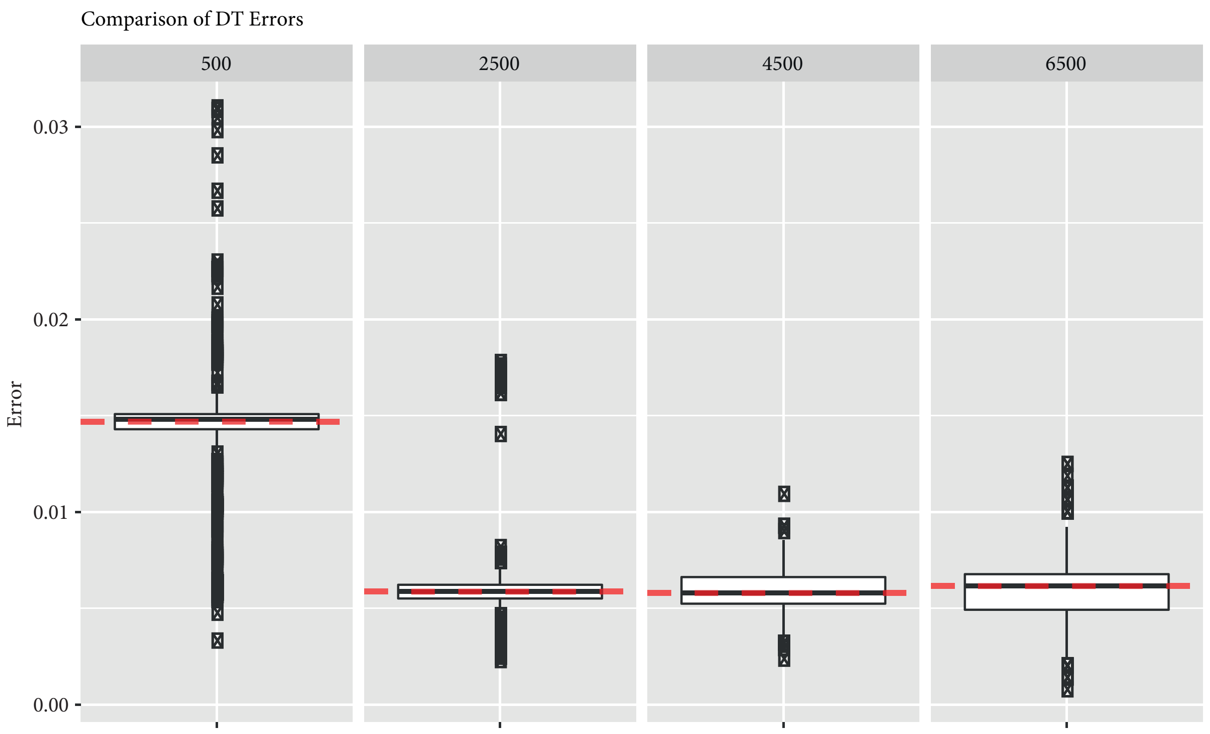

Figure 6. Comparison of decision tree error rates with respect to sample sizes and sampling method.

The first thing to notice in this figure is the variance of the classification errors in random samples. This points out the fact that the classification error obtained over a randomly generated training set is subject to a significant variance including numerous outliers. It should also be obvious that the test set is dependent on the selected training set. This indeed brings additional uncertainty regarding the resulting classification score of the entire learning process. Finally, our results clearly demonstrates that the classification error of OS almost perfectly position at the median of the error distribution of the random samples. This is a clear indication that OS is neither positively nor negatively biased. Lastly, we report that the computation time of the MIP model is around a few minutes for all instances solved.

4. Conclusion

In this paper, as an alternative to using a random sampling approach, we propose a novel method for partitioning a dataset into training and test subsets for a machine learning study. The proposed method is based on the idea of finding training and test sets whose distributions best fit to that of the original dataset. This eventually enables us to eliminate the noise introduced by standard random sampling approaches in selecting training and test sets. To this end, we employed a simple yet effective distance metric referred to as histogram matching and embed it to an optimization problem. We developed the mixed integer programming formulation of the problem.

We then conducted a numerical study to investigate the efficiency of the proposed method and observe its impact on an actual machine learning study. The numerical study demonstrated that the optimized training set establishes a benchmark for the given dataset and training set size. Also, our results clearly revealed that the optimized sets found by the proposed method have far less distribution matching error with respect to histogram matching. Furthermore, we also compared the prediction accuracy of a decision tree trained by random samples and optimized set found by our method. The former is highly dispersed with varying training sets and subject to a significant level of uncertainty. The latter, on the other hand, uniquely positions itself onto that of randomly generated sets corresponding to the median value. This indicates that our method introduces neither positive nor negative bias in prediction accuracy. As such, the proposed method clearly enhances the robustness of machine learning algorithms by eliminating the bias induced due to random sampling.

There are many avenues for further research. It is important to extend the proposed approach onto datasets where features are not necessarily categorical. Also, it would be interesting to investigate the practical extent of the proposed approach on a very comprehensive numerical study considering variety of datasets. It would also be of interest to investigate computationally efficient solution methods for the proposed approach to be able to work with big data.

Acknowledgment

We acknowledge that all work done, including the idea development, computer implementation, numerical studies and authoring the paper, were shared equally by the authors.

References

[1] Turing AM. Computing machinery and intelligence. Mind 1950; 59: 433-460.

[2] Guyon I, Elisseeff A. An introduction to variable and feature selection. Journal of Machine Learning Research 2003; 3: 1157-1182.

[3] Battiti R. Using mutual information for selecting features in supervised neural net learning. IEEE Transactions on Neural Networks 1994; 5: 537-550.

[4] Forman G. An extensive empirical study of feature selection metrics for text classification. Journal of Machine Learning Research 2003; 3: 1289-1305.

[5] Peng H, Long F, Ding C. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. IEEE Transactions on Pattern Analysis and Machine Intelligence 2005; 27: 1226-1238.

[6] Kohavi R, John GH. Wrappers for feature subset selection. Artificial Intelligence 1997; 97: 273-324.

[7] Narendra PM, Fukunaga K. A branch and bound algorithm for feature subset selection. IEEE Transactions on Computers 1977; 9: 917-922.

[8] Pudil P, Novovicova J, Kittler J. Floating search methods in feature selection. Pattern Recognition Letters 1994; 15: 1119-1125.

[9] Reunanen J. Overfitting in making comparisons between variable selection methods. Journal of Machine Learning Research 2003; 3: 1371-1382.

[10] Chandrashekar G, Sahin F. A survey on feature selection methods. Computers & Electrical Engineering 2014; 40: 16-28.

[11] Garcia S, Derrac J, Cano J, Herrera F. Prototype selection for nearest neighbor classification: Taxonomy and empirical study. IEEE Transactions On Pattern Analysis and Machine Intelligence 2012; 34: 417-435.

[12] Pkekalska E, Duin RPW, Paclik P. Prototype selection for dissimilarity-based classifiers. Pattern Recognition 2006; 39: 189-208.

[13] Garcia S, Cano J, Herrera F. A memetic algorithm for evolutionary prototype selection: A scaling up approach. Pattern Recognition 2008; 41: 2693-2709.

[14] Arnaiz-Gonzalez A, Diez-Pastor JF, Rodriguez JJ, Garcia-Osorio C. Instance selection of linear complexity for big data. Knowledge-Based Systems 2016; 107: 83-95.

[15] Song Y, Liang J, Lu J, Zhao X. An efficient instance selection algorithm for k nearest neighbor regression. Neurocomputing 2017; 251: 26-34.

[16] Li Y, Hu Z, Cai Y, Zhang W. Support vector based prototype selection method for nearest neighbor rules. In: International Conference on Natural Computation; 2005; Berlin, Germany: Springer 528-535.

[17] Liu Chuan, Wang W, Wang M, Lv F, Konan M. An efficient instance selection algorithm to reconstruct training set for support vector machine. Knowledge-Based Systems 2017; 116: 58-73.

[18] Srisawat A, Phienthrakul T, Kijsirikul B. SV-kNNC: An algorithm for improving the efficiency of k-nearest neighbor. In: Pacific Rim International Conference on Artificial Intelligence; 2006; Berlin, Germany: Springer 975-979.

[19] Brighton H, Mellish C. Advances in instance selection for instance-based learning algorithms. Data Mining and Knowledge Discovery 2002; 6: 153-172.

[20] Riquelme J, Aguilar-Ruiz J, Toro M. Finding representative patterns with ordered projections. Pattern Recognition 2003; 36: 1009-1018.

[21] Raicharoen T, Lursinsap C. A divide-and-conquer approach to the pairwise opposite class-nearest neighbor (POC-NN) algorithm. Pattern Recognition Letters 2005; 26: 1554-1567.

[22] Silva DA, Souza LC, Motta G. An instance selection method for large datasets based on markov geometric diffusion. Data & Knowledge Engineering 2016; 101: 24-41.

[23] Ashfaq RAR, He Y, Chen D. Toward an efficient fuzziness based instance selection methodology for intrusion detection system. International Journal of Machine Learning and Cybernetics 2017; 8: 767-1776.