IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 23, 2020, accepted October 25, 2020, date of publication October 30, 2020,
date of current version November 12, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3034955

On the Effectiveness of Discretizing Quantitative Attributes in Linear Classifiers

NAYYAR A. ZAIDI[1], YANG DU[2], AND GEOFFREY I. WEBB[2], (Fellow, IEEE)
[1]Faculty of Science, Engineering and Built Environment, School of Information Technology, Deakin University, Geelong, VIC 3220, Australia
[2]Faculty of Information Technology, Monash University, Melbourne, VIC 3800, Australia

Corresponding author: Nayyar A. Zaidi (nayyar.zaidi@deakin.edu.au)

ABSTRACT Linear models in machine learning are extremely computational efficient but they have high representation bias due to non-linear nature of many real-world datasets. In this article, we show that this representation bias can be greatly reduced by discretization. Discretization is a common procedure in machine learning that is used to convert a quantitative attribute into a qualitative one. It is often motivated by the limitation of some learners to handle qualitative data. Since discretization looses information (as fewer distinctions among instances are possible using discretized data relative to undiscretized data) – where discretization is not essential, it might appear desirable to avoid it, and typically, it is avoided. However, in the past, it has been shown that discretization can leads to superior performance on generative linear models, e.g., naive Bayes. This motivates a systematic study of the effects of discretizing quantitative attributes for discriminative linear models, as well. In this article, we demonstrate that, contrary to prevalent belief, discretization of quantitative attributes, for discriminative linear models, is a beneficial pre-processing step, as it leads to far superior classification performance, especially on bigger datasets, and surprisingly, much better convergence, which leads to better training time. We substantiate our claims with an empirical study on 52 benchmark datasets, using three linear models optimizing different objective functions.

INDEX TERMS Discretization, classification, logistic regression, support vector classifier, artificial neuron, big datasets, bias-variance analysis.

I. INTRODUCTION

Linear models in machine learning are popular due to their simplicity and computational efficiency. Generative linear models such as naive Bayes, and discriminative linear models such as Logistic Regression, are commonly used as baseline models during model evaluation phase [1]. In some real world problems, these simple linear models perform very well in comparison to sophisticated non-linear models such as Factorization Machines, Bayesian Networks, Artificial Neural Networks, Gradient Boosted Decision Trees, etc. However, due to their simplicity, these linear models do have an inherent weakness which stems from their high *representation bias* [2]. The representation bias, also called, its *hypothesis language bias* [3], can be defined as the minimum loss of any model in the space of models available to the learner. It is clearly desirable in the general case to use a space of models with minimum representation bias, for any given problem. This is one reason, non-linear models which have inherent (explicit or implicit) feature engineering process leads to superior performance on many real-world datasets.

Discretization is a typical pre-processing step in many machine learning algorithms that is used to convert a Quantitative attribute into Qualitative one [4], [5]. In this article, we argue that high representation bias of typical linear models on real-world problems, can be reduced by discretization. Typically, the representation-bias reduction is done through either manual or automated feature engineering. Manual feature engineering is the process of creating new features from the original features. Each non-linear machine learning model has its own inherent feature engineering phase, which could be explicit, e.g., in case of higher-order Logistic Regression [6], k-Dependence Bayesian Estimators [7], etc. or it could be implicit, e.g., Artificial Neural Networks have layers of dense, convolution, recurrent layers, which effectively engineer features to be fed later to the linear part of the network [8], [9]. This is illustrated in Figure 1, where two schemes for reducing the representation bias of a model are

The associate editor coordinating the review of this manuscript and approving it for publication was Shunfeng Cheng.

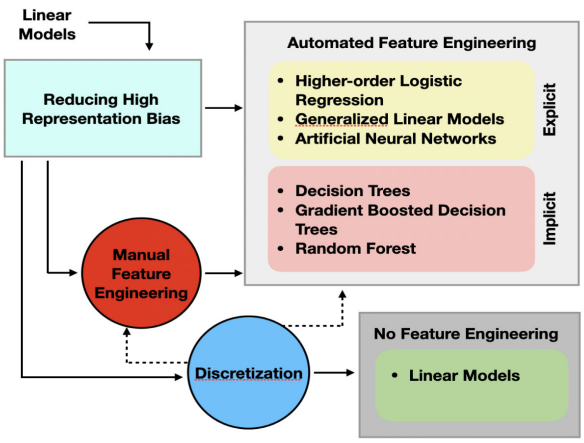


FIGURE 1. Illustration of the two schemes of reducing representation bias of simple linear models – a) Feature Engineering, b) Discretization.

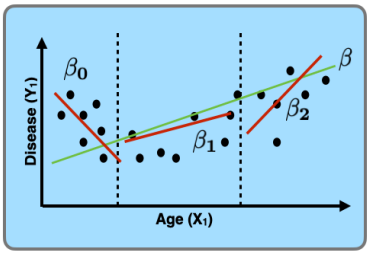


FIGURE 2. Contrived example demonstrating the power of discretization for reducing representation bias of a linear model.

shown. Note, we do not propose discretization as an alternative to feature engineering – rather, we present it as a process which share its goal of reducing representation bias of the model, with feature engineering process. We believe that this role of discretization has somewhat been mis-understood or under-appreciated – as practice of discretizing quantitative attributes is not common in practical machine learning, due to its apparent data loss perspective.

Consider the simple contrived univariate regression example in Figure 2. The dependent variable Y_1 is plotted on the y-axis whereas the independent variable X_1 is on x-axis. As can be seen that there is a non-linear trend in the data as the extent of the disease is more prevalent in early and late age group. A linear model, will learn just one parameter β – which corresponds to the slope of the plane. However, if we discretize the independent variable, now our linear model is forced to learn three different parameters, i.e., $\beta_1, \beta_2, \beta_3$ – which can lead to modelling this non-linear trend. We have demonstrated this in regression settings (e.g., linear regression), but the underlying idea is equally applicable to classification (e.g, logistic regression) problems, that we address in this article.

Let us now consider another example, demonstrated in Figure 3. A linear model is trained on a synthetically generated 2-D data, where one class (Class B) is surrounded by the second class (Class A). As can be seen from Figure 3(a) – a linear model does not have the power to capture the non-linear boundary. In Figure 3(b), we discretize the data using 4-bin equal frequency discretization – followed by the application of the same linear model. It is surprising to see that the resulting discretization has produced features in which data can be separated by even linear models. Is this trend ubiquitous, i.e, is discretization a panacea for linear models when handling non-linear data? We believe the answer is No. We give another example of a dataset in Figure 4, where linear model with or without discretization is not very effective. We will discuss soon, that on datasets such as those shown in Figure 4 – traditional feature engineering either manual or automated will be more effective, which we discuss later in this section. Now, based on above examples, let us state two important observations, based on which we would like to motivate this work:

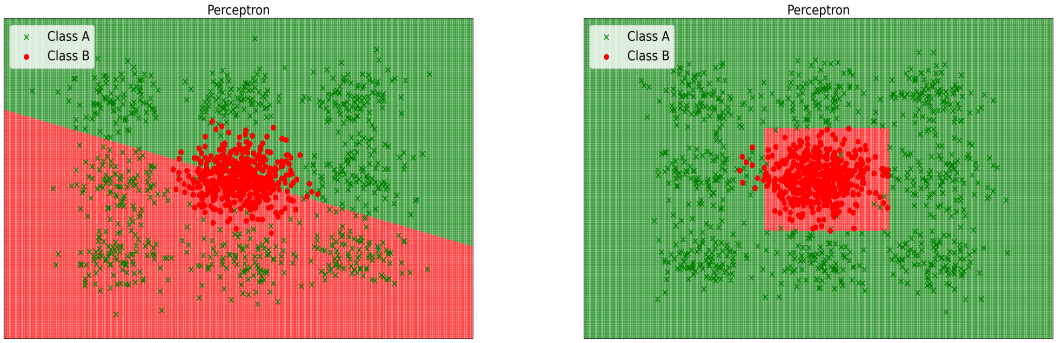
- First, a linear classifier with discretization is not linear with respect to the original data anymore.
- Contrary to the predominant perception of discretization as an information loss process – an alternative perspective on discretization is that, a linear classifier with discretization has reduced representation bias, which consequently results in learning a model with much reduced error. The motivation for writing this article is to highlight this perspective.

The contributions of this work are two-fold:

- First, through a systematic evaluation on standard datasets, we show that discretization can greatly reduce the error of typical discriminative linear models such as those optimizing Conditional Log-Likelihood (CLL), Hinge Loss (HL) and Mean-Square-Error (MSE) objective functions. As we discussed earlier, this is contrary to popular belief, and hence a significant finding.
- Second, we show that discretization based models have much better convergence profiles, which lend them faster training time, which is greatly desirable when learning from large quantities of data.

Note that we do not claim that discretization is an alternative to feature engineering (or any other form of feature transformation – note, all feature transformations, either linear or non-linear are subsumed in our use of the term *Feature Engineering*). For instance, an example of a case where discretization is not very useful is demonstrated by the dataset shown in Figure 4. It can be seen that for this dataset, either manual feature engineering or automated feature engineering capability of e.g., tree-based methods or of Artificial Neural Networks will be much more effective. This is demonstrated in Figure 5.

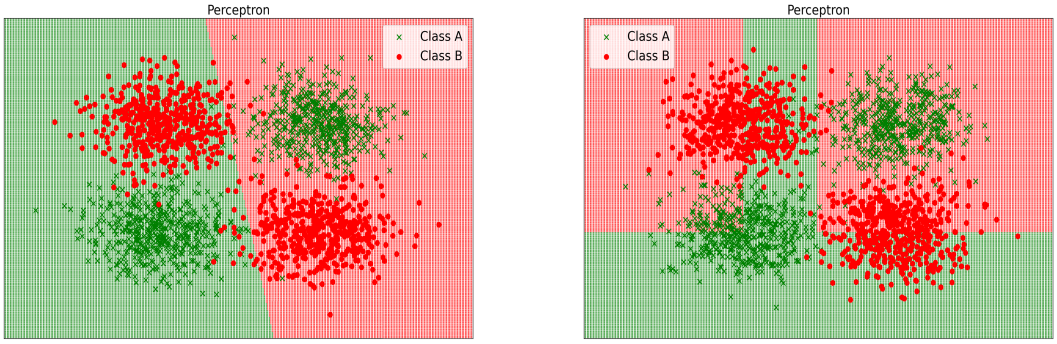
We must also state that this is not a study of comparative analysis of different discretization schemes or that of different discriminative linear model. There are several work that already discuss these two topics in some detail [10], [11]. Finally, we conjecture, that discretization can be taken as a complimentary technique to feature engineering. And, there is a strong case to study the effectiveness of discretization for even non-linear models (depicted as two dotted arrows



(a) Perceptron on original data

(b) Perceptron trained on discretized data. 4-bin equal frequency discretization is used.

FIGURE 3. **[Figure is seen best in color] Illustration of the effectiveness of a discretization. Perceptron is chosen as a linear model. It is a model of not so practical importance but chosen here as a representative of linear models. Similar results are to be expected with Logistic Regression and Support Vector Classifier. Equal frequency distribution is also chosen as representative. The results are generalizable to other forms of discretization. It can be seen that on this non-linear data distribution, discretization has been quite effective.**



(a) Perceptron on original data

(b) Perceptron trained on discretized data. 4-bin equal frequency discretization is used.

FIGURE 4. **Comparison of decision boundaries with and without discretization on a simple two-dimensional X-OR problem. It can be seen that on this non-linear data distribution, discretization has not been very helpful.**

in Figure 1), especially for Artificial Neural Networks, with the emergence of deep learning [12].

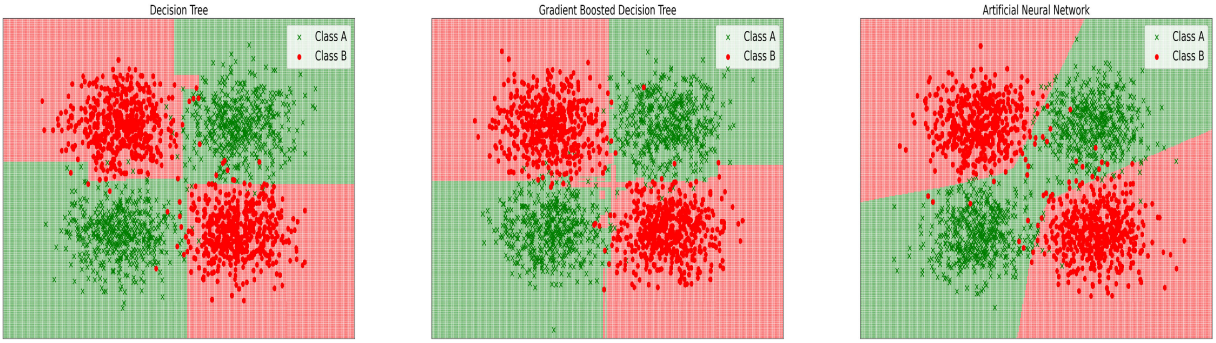
The rest of this article is organized as follows. Some preliminary background and terminology is given in Section II-A. We discuss discretization in general in Section II-B. Linear classifiers based on Conditional Log-Likelihood (CLL) Hinge Loss (HL) and Mean-square-error loss are discussed in Sections II-D, II-E and II-F respectively. Optimization strategies for training these linear classifiers are discussed in Section II-H. An overview of related work is given in Section III. Experimental results are given in Section IV. We conclude in Section V with pointers to directions for future work.

II. BACKGROUND

A. TERMINOLOGY

In machine learning and data mining research, there exists variation in the terminology when it comes to characterizing the nature of an attribute (or feature). For example, 'continuous vs. discrete', 'numeric vs. categorical' and 'quantitative vs. qualitative'. We believe that the 'quantitative vs. qualitative' distinction is best suited for our study in this article and hence, this is used throughout the paper.

Qualitative attributes are the attributes on which arithmetic operations can not be applied. The values of a qualitative attribute can be placed or categorized in distinct categories. Sometimes there exist a meaningful rank among these categories, resulting in distinction of ordinal and nominal among quantitative attributes. For example, *Student Grade: {HD, D, C, P, F}* and *Pool Depth: {Very Deep, Deep, Shallow}* are ordinal attributes, while *Marital Status: {Married, Never-married, Divorced, Widow, Widower}* and *Nationality: {Australian, American, British}* are nominal attributes. Quantitative attributes, on the hand, are the attributes on which arithmetic operations can be applied. They can be both discrete and continuous. For example, *Number of Chil-*



(a) Decision Tree (b) Gradient Boosted Decision Tree (c) Artificial Neural Network

FIGURE 5. Comparison of decision boundaries obtained with Decision Tree, with maximum depth of 6 (Left), Gradient Boosted Decision Trees, with maximum depth of 2 (Middle), and Artificial Neural Network, with 1 hidden layer containing 30 nodes ran for 100 iterations maximum with Adam optimizer (Right) on original data on a simple two-dimensional X-OR problem.

TABLE 1. List of symbols use in this work.

Notation	Description
I	Number of qualitative attributes
K	Number of quantitative attributes
n	Total number of attributes, $n = I + K$
N	Number of data points in $\mathcal{D}$
$\mathcal{D} = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$	Data consisting of N objects
$\mathcal{L} = \{y^{(1)}, \ldots, y^{(N)}\}$	Labels of data points in $\mathcal{D}$
$\tilde{\mathrm{P}}(e)$	Actual probability of event (e)
$\mathrm{P}(e)$	Probability of event e
$\mathrm{P}(e \mid g)$	Conditional probability of event e given g
$\mathbf{x} = \langle x_0, x_1, \ldots, x_n \rangle$	An object (n-dimensional vector) and $\mathbf{x} \in \mathcal{D}$
Y	Random variable associated with class label
y	$y \in Y$. Class label for object. Same as x_0
C	$\lvert Y \rvert$, Number of classes
X_i	Random variable associated with qualitative attribute i
x_i	Actual value that X_i takes.
$\lvert X_i \rvert$	(Applicable only to qualitative attributes) Number of values of attribute X_i
β	LR parameter vector to be optimized
$\beta_{y,i}$	LR parameter associated with quantitative attribute i for class y
$\beta_{y,k,j}$	LR parameter associated with qualitative attribute k for class y taking value j
$\beta_{y,0}$	LR intercept term for class y.
λ	Regularization parameter

dren is a discrete-quantitative attribute (values determined by counting), whereas *Temperature* is a continuous-quantitative attribute (values determined by measuring).

We use the term feature, attribute and category interchangeably in the paper. The list of various symbols used in this work is given in Table 1.

B. DISCRETIZATION

Discretization is a common process in machine learning that is used to convert a quantitative attribute into a qualitative attribute [5], [13]. The need for discretization originates from the fact that some classifiers can only handle qualitative attributes or operate better with qualitative attributes, e.g., naive Bayes or Bayesian Networks. The process of discretization involves finding cut-points within the range of the quantitative attribute and to group values into intervals based on these cut-points. This removes the ability to distinguish between data points falling in the same interval. Therefore, discretization entails information loss.

Discretization methods can be categorized into two categories: *Supervised* and *Un-supervised*. In the unsupervised case, class information is not used during cut-point determination process. Popular approaches are equal-frequency and equal-width discretization. Equal-width discretization (EWD) divides the quantitative attribute's range (maximum value x_i^{max} and minimum value x_i^{min}) into k equal-width intervals where k is provided by the user. Each interval will have a width of:

$$w = \frac{x_i^{max} - x_i^{min}}{k}.$$

Equal-frequency discretization (EFD), on the other hand, divides the sorted values of a quantitative attribute such that each interval has approximately k number of data points. Each interval will contain N/k data points. It is also important that data points with identical value are placed in the same interval, therefore, in practice, each interval will have slightly different number of data points. Choosing EWD or EFD and the number of bins is problem specific and can have a huge impact on the overall performance of any model. Of course, choosing a large k will result in less information loss, but can result in over-fitting on small datasets.

Supervised discretization methods, on the other hand, utilize the class information of the data point to better define the cut-points. For example, state-of-the-art discretization technique Entropy-Minimization Discretization (EMD) sorts the quantitative attribute's values and then finds the cut-point such the information gain is maximized across the splits [14]. The technique is applied recursively on the successive splits and the minimum-description-length (MDL) criterion is used to determine when to stop splitting.

Another distinction can be made among discretization methods in terms of either lazy or eager nature of discretization. The discretization algorithms discussed so far are eager as the discretization happens at training time. Lazy methods are computationally intensive as they delay discretization until classification time. When a test instance has arrived, lazy methods of discretization go through the entire data to find the cut-point [15], [16].

C. TOWARDS QUALITATIVE FEATURES REPRESENTATION

Many models including Artificial Neural Networks can not handle qualitative attributes, and a common strategy is to use one-hot-encoding to convert qualitative features to quantitative features (each taking either 0 or 1 values). The resulting representation is storage inefficient, and also semantically useless. The resulting binary quantitative attributes are then fed to the model. With the emergence of *Representation Learning*, and especially with the demonstrated success of methods such as word2vec, Glove, etc. [17], [18], for providing dense and meaningful category (or feature) representation – there is a widespread belief in community that categorical or qualitative features, if represented properly, can be extremely useful. Some preliminary studies of obtaining categorical embeddings have already been shown to be quite effective [19]–[21], however, a systematic study of converting quantitative attributes into qualitative attributes and then learning to obtain an embedding for each category, which later are to be fed to the model is yet to be seen. We argue, that discretization will play an important role in any such study.

D. LINEAR CLASSIFIER - CLL

One of the best example of a linear model is Logistic Regression (LR). It is the workhorse of Statistics community and one of the state of the art classifier. It optimizes the conditional log-likelihood (CLL) as its objective function, which can be defined as:

$$\text{CLL}(\beta) = \sum_{l=1}^{N} \log P(y^{(l)} \mid x^{(l)}), \tag{1}$$

where

$$P(y^{(l)} \mid x^{(l)}) = \frac{\exp(\beta_{y,0} + \beta_y^T x^{(l)})}{\sum_{c=1}^{C} \exp(\beta_{y,0} + \beta_c^T x^{(l)})}. \tag{2}$$

The term $\beta_{y,0} + \beta_y^T x^{(l)}$ is expanded as:

$$\beta_{y,0} x_0^{(l)} + \beta_{y,1} x_1^{(l)} + \beta_{y,2} x_2^{(l)} + \cdots + \beta_{y,n} x_n^{(l)},$$

where x_0 can be assumed to be 1 for all data points. Since the objective function as defined in Equation 1, is linear in x, it is a linear classifier. Equation 2 leads to a multi-class softmax objective function. Since, a set of parameters are learned for each class, we have made this distinction explicit with subscript y in parameter notation, that is, $\beta_{y,j}$ denotes a parameter for class y and attribute j.

Typically, an LR expects all input attributes to be quantitative and minimizes the negative of the CLL known as negative log-likelihood (NLL), which is defined as:

$$\text{NLL}(\beta) = -\sum_{l=1}^{N} \left(\left(\beta_{y,0} + \beta_y^T x^{(l)} \right) - \log \left(\sum_{c=1}^{C} \exp(\beta_{c,0} + \beta_c^T x^{(l)}) \right) \right). \tag{3}$$

Note, in the following, for simplicity, we will drop the superscript (l) notation.[1] Nonetheless, optimizing a standard LR with NLL based either on Equation 3 or Equation 4 requires substantial input manipulation, i.e., appending 1 to all data points and then, converting qualitative attributes using one-hot-encoding. For example, a qualitative attribute X_j taking values $\{a, b, c\}$, will be converted into three attributes X_j, X_{j+1}, X_{j+2}, each taking values either 0 or 1. An alternative to manipulating the input is to modify the model and optimize the following objective function instead:

$$\begin{aligned}\text{NLL}(\beta) &= -\sum_{l=1}^{N} \left(\left(\beta_{y,0} + \sum_{i=1}^{I} \beta_{y,i} x_i + \sum_{k=1}^{K} \sum_{j=1}^{|X_k|} \beta_{y,k,j} \mathbf{1}_{X_k = x_j} \right) \right. \\ &\quad \left. - \log \left(\sum_{c=1}^{C} \exp(\beta_{c,0} + \sum_{i=1}^{I} \beta_{c,i} x_i + \sum_{k=1}^{K} \sum_{j=1}^{|X_k|} \beta_{c,k,j} \mathbf{1}_{X_k = x_j}) \right) \right).\end{aligned} \tag{5}$$

Note that the models expressed in Equation 3 and 5 are exactly equivalent and will lead to the same results. The only difference is that the model in Equation 3 requires converting all qualitative attributes into quantitative ones using one-hot-encoding, whereas the models in Equation 5 do not. Equation 5 can be simplified even further – for datasets with only qualitative attributes (note, this is the case at our hand, as all quantitative attributes will be converted into qualitative ones, after discretization), and including only terms that are not canceled out, we have:

$$\begin{aligned}\text{NLL}(\beta) &= -\sum_{l=1}^{N} \left(\beta_{y,0} + \sum_{k=1}^{K} \beta_{y,k,j} \mathbf{1}_{X_k = x_j, Y = y} \right. \\ &\quad \left. - \log \left(\sum_{c=1}^{C} \exp(\beta_{c,0} + \sum_{k=1}^{K} \beta_{c,k,j} \mathbf{1}_{X_k = x_j, Y = c}) \right) \right).\end{aligned} \tag{6}$$

[1] It should be noted that many software libraries for multi-class LR are either based on implementing multi-class (softmax) objective function of Equation 3 or they optimize a more simpler binary objective function of the following form:

$$\text{NLL}(\beta) = \sum_{l=1}^{N} \left(\log \left(1 + \exp(\beta_0 + \beta^T x^{(l)}) \right) \right), \tag{4}$$

and solve a one-versus-all classification problem. Note that in the case of binary classifiers, there is only one set of parameters for the two classes as oppose to C set of parameter that needed to be optimized for the softmax case. At classification time, one needs to apply C different trained LR classifiers and choose one with the highest probability.

The typical process when using LR model is to convert qualitative attributes into quantitative ones and optimize the model of Equation 3. Instead, in this article, we argue that one should convert quantitative attributes into qualitative ones using discretization methods as discussed in Section II-B and use the model of Equation 6.

It can be seen that with Equation 3, the number of parameters optimized are: $(C-1)+(C-1)n$. Whereas, with Equation 6, $(C-1)+(C-1)\sum_{i=1}^{n}|X_i|$ parameters are optimized. Since the two models do not have equal number of parameters, as well as the scale of each feature is different – we claim that this will result in different training time, speed and rate of convergence and of course, classifications.

E. LINEAR CLASSIFIER-HINGE LOSS

Hinge Loss (HL) is widely used as an alternative to CLL and has been the basis of Support Vector Machines. A classifier optimizing either a Hinge Loss objective function or its variant is a linear classifier and is known as the Support Vector Classifier (SVC). Here we define L2-Loss HL as:

$$\text{HL}(\beta) = \sum_{l=1}^{N} \max(0, 1 - y\beta^T x)^2. \tag{7}$$

An alternative is L1-Loss HL which is equal to: $\sum_{l=1}^{N}\max(0, 1-y\beta^T x)$. In this work, we will focus only on the L2-Loss. In practice, a penalty term is also added for regularizing the objective function as:

$$\text{HL}(\beta) = \frac{1}{2}||\beta^T\beta||^2 + \lambda\sum_{l=1}^{N}(\max(0, 1 - y\beta^T x))^2, \tag{8}$$

where λ is the regularization parameter. We will discuss the gradient and Hessian of this objective function later in Section II-H.

F. LINEAR CLASSIFIER-MEAN-SQUARE-ERROR

Another linear classifier is based on optimizing the Mean-Square-Error (MSE) objective function and is defined as:

$$\text{MSE}(\beta) = \frac{1}{2}\sum_{l=1}^{N}\sum_{c=1}^{C}(\tilde{P}(c\,|\,x) - P(c\,|\,x))^2, \tag{9}$$

where $P(c\,|\,x)$ is given in Equation 2 and $\tilde{P}(c\,|\,x)$ is the actual probability of class c given data instance x. This will be a vector of size C with all zeros except at the location of the label of x, where it will be 1 (assuming there are no duplicate data points in the dataset). The objective function of Equation 9 is similar to that optimized by artificial neural-networks (ANN). However, in ANN, $P(c\,|\,x)$ is defined in terms of multiple layers. We can interpret Equation 9 as a zero-layer ANN.

G. LINEAR CLASSIFIER-OTHER LOSSES

We have constrained ourselves to three loss functions as described in previous sections – Negative Log-Likelihood, Hinge Loss and Mean-Square-Error. It must be mentioned that many other loss functions exist, which leads to different variants of linear models. For example, another variant of Hinge Loss – Categorical Hinge Loss, is quite popular. There are several alternative probabilistic losses to NLL, such as, KL Divergence loss, poisson loss, sparse categorical cross-entropy loss etc. Several other variants of mean-square-loss exists as well, e.g., Huber loss, mean absolute percentage error, mean squared log error, cosine similarity etc. As we mentioned earlier, these different losses will lead to different linear models. An evaluation of all these loss functions is beyond the scope of this work. We have chosen three loss functions as the representative of these various loss functions, to study the effect of discretization on the performance of linear models.

H. OPTIMIZATION

There is no closed form solution to optimizing the negative log-likelihood, hinge loss and mean-square-error objective function, and, therefore, one has to resort to iterative minimization procedures, which could be either first-order or second-order, depending-upon if the Hessian is used for tuning the step size. It is well-known that the quality of solution found using gradient-based optimization methods is greatly affected by the scaling of the axis, as well as the number of dimensions. Therefore, the scale of the input features should be taken rather seriously. We conjecture, that any decision to discretize attributes will effect the quality of convergence, quality of global optimum solution found, computational efficiency and accuracy of the optimization results.

In iterative optimization the procedure generates a sequence $\{\beta^k\}_{k=1}^{\infty}$ converging to an optimal solution. At every iteration k, the following update is made: $\beta^{k+1} = \beta^k + s^k$, where s^k is the search direction vector. The following equation plays the pivotal role as it holds the key to obtain s^k by solving a system of linear equations:

$$\nabla^2 f(\beta^k)s^k = -\nabla f(\beta^k), \tag{10}$$

where f is the objective function that we are optimizing. There are two very important issues that must be addressed when solving for search direction vector using Equation 10 [22]. First, it can be infeasible to explicitly compute and store the Hessian, especially on high-dimensional data. Second, the solution obtained using Equation 10, does not guarantee convergence. There are two main strategies for addressing these issues, leading to distinction of first-order and second-order optimization that we discuss in the following two sections.

1) FIRST ORDER METHODS

Consider $\nabla^2 f(\beta^k)$ to be an identity matrix in Equation 10 – in this case, $s^k = -\nabla f(\beta^k)$. This leads to a family of algorithms known as first-order methods such as Gradient Descent, Coordinate Descent, etc. The step-size, however, has to be tuned manually and plays a critical role in determining the quality of the solution Also, the algorithm might take much longer to converge, and could be slow. These issues

are addressed by either mini-batch or stochastic variants of these methods, which inadvertently leads to high-variance problem. Several techniques to reduce the variance has been proposed, which has to led to methods such as Adagrad [23], RMSProp [24], Adam [25]. These methods have become rather de-facto standard in typical machine learning and deep learning rsearch. Note, the step-size tuning of these algorithms is still an open research question, and can significantly impact the optimization. Even though, these methods are standard now in state of the art libraries such as Tensor Flow, Torch, etc. – we have not used them in this work. Instead, we rely on approximate second-order methods as discussed in the following section.

2) SECOND ORDER METHODS

Instead of considering $\nabla^2 f(\beta^k)$ to be an identity matrix in Equation 10, as discussed in the previous section, we can aim not to compute $\nabla^2 f(\beta^k)$ directly, but approximate it from the information present in $\nabla f(\beta^k)$ instead. This property is useful for large scale problems where we cannot store the Hessian matrix. This leads to approximate second-order methods known as quasi-Newton algorithms, for example, L-BFGS which, is considered to be the most efficient algorithm (de-facto standard) for training LR.

Another possibility of solving Equation 10, might be to use standard 'direct algorithms' for solving a system of linear equations such as Gaussian elimination to solve for s^k. Several variants that are popular in this category for optimizing LR related models is that of 'Truncated Newton method' [26] – also known as TRON, conjugate gradient, etc. [27].

3) OPTIMIZATION PROFILE

It should be noted that various optimization methods discussed so far, differ in terms of the speed-of-convergence, cost-per-iteration, iterations-to-convergence, etc. For example, first-order optimization method such as Coordinate Descent updates one component of β at every iteration, so the cost-per-iteration is very low, but iterations-to-convergence will be very high. On the other hand, second-order methods such as quasi-Newton method, will have high cost-per-iteration, but very low number of iterations-to-convergence. It is also important to note that all optimization methods are all affected by the scaling of the axis. Therefore, scaling quantitative attributes or converting quantitative into qualitative attributes will effect the speed and the quality of the convergence:

4) DERIVING GRADIENTS AND HESSIAN

In the following, we will define the gradient and Hessian of the three objective functions – conditional log-likelihood, hinge loss and mean square error. Note, we only define the gradient and the Hessian for qualitative attributes here. For softmax CLL, $\nabla f(\beta^k)$ and $\nabla^2 f(\beta^k)$ can be written as:

$$\frac{\partial \text{NLL}(\beta)}{\partial \beta_{y',i}} = \sum_{l=1}^{N} (\mathbf{1}_{y=y'} - P(y'|x))x_i,$$

$$\frac{\partial^2 \text{NLL}(\beta)}{\partial \beta_{y',i} \partial \beta_{y'',j}} = -\sum_{l=1}^{N} (\mathbf{1}_{y'=y''} - P(y'|x))P(y''|x)x_i x_j.$$

For Hinge-loss, the (sub-) gradients can be written as:

$$\frac{\partial \text{HL}(\beta)}{\partial \beta_i} = \beta_i + 2C \sum_{l=1}^{N'} (y\beta^T x - 1).y\, x_i,$$

where N', are the instances for which $y\beta^T x < 1$ is true. Similarly, for HL, the (sub-) Hessian can be written as:

$$\frac{\partial^2 \text{HL}(\beta)}{\partial \beta_i \partial \beta_j} = \mathbf{1}_{i=j} + 2C \sum_{l=1}^{N'} x_i x_j$$

For MSE, one can write the gradients as:

$$\frac{\partial \text{MSE}(\beta)}{\partial \beta_{j,x_j,k}} = \sum_{l=1}^{N} \sum_{c=1}^{C} (\mathbf{1}_{y=c} - P(c|x))(\mathbf{1}_{k=c} - P(k|x))x_i,$$

and the Hessian for MSE can be written as shown in Equation 11, as shown at the bottom of the page.

5) ON THE CHOICE OF OPTIMIZATION

First-order methods with automated gradient computation are new de-facto standards in emerging deep learning research. Also, gradient calculation such as forward-mode Auto-diff, reverse-mode Auto-diff, etc. are quite effective – so even the gradient formula does not need to be specified. This greatly simplifies model building process, but also increases many number of hyper-parameters in the system, e.g., step size, optimization method, gradient computation method, stopping criterion, etc. In this work, we have constrained ourselves to approximate second order-methods such as L-BFGS and Tron, to keep the optimization free from any hyper-parameter tuning, and to focus entirely on the discretization and input manipulation part.

$$\begin{aligned}\frac{\partial^2 \text{MSE}(\beta)}{\partial \beta_{j,x_j,k} \partial \beta_{j',x_{j'},k'}} = -\sum_{l=1}^{N} \big[& (-1)(\mathbf{1}_{k'=c} - P(k'|x))P(c|x)\mathbf{1}_{j=j'} \\ & + (\mathbf{1}_{y=c} - P(c|x))(-1)(\mathbf{1}_{k'=k} - P(k'|x))P(k|x)\mathbf{1}_{j=j'}P(c|x) \\ & + (\mathbf{1}_{y=c} - P(c|x))(\mathbf{1}_{k=c} - P(k|x))(\mathbf{1}_{k'=c} - P(k'|x))P(c|x)\mathbf{1}_{j=j'} \big] x_i x_j. \end{aligned} \quad (11)$$

III. RELATED WORK

Discretization is often motivated by a need to adapt data for a model that cannot handle quantitative attributes. In Statistics and many of its related and applied branches (such as epidemiology, medical research and consumer marketing), it goes by names of 'dichotomization' and 'categorization' (where the two techniques differ as the former splits the measurement scale into two while the later can have more than two categories) – and has been examined in many studies [28]–[30]. However, in most of these studies a majority opinion is against the use of dichotomization – and for categorization, it is advised to be used with caution. The main reason cited for this is that dichotomization and categorization lead to information loss since the variability among the members of the group is subsumed. For example, [31] write:

> ...Firstly, much information is lost, so the statistical power to detect a relation between variable and patient outcome is reduced ...and considerable variability may be subsumed within each group. Individuals close to but on opposite sides of cut-point are characterized as being very different rather than very similar ...

In practical machine learning, the common practice is to discretize an attribute only if necessary (i.e., if a model expects categorical attributes). An exception is for Bayesian classifiers, where it is common practice to discretize numeric attributes [32].

The ambivalence towards discretization is understandable. Obviously, the quality (and sometime quantity) of data is the key to training accurate models and hence getting good results. In many cases, the data are the result of costly and time-consuming efforts (for example in breast cancer research where there are several stake-holders involved just to obtain a few attributes of the data). Losing some of the data (or more precisely, losing some distinction among the instances) due to discretization should be undesirable. However, in existing research, a number of motivations for discretization have been put forward:

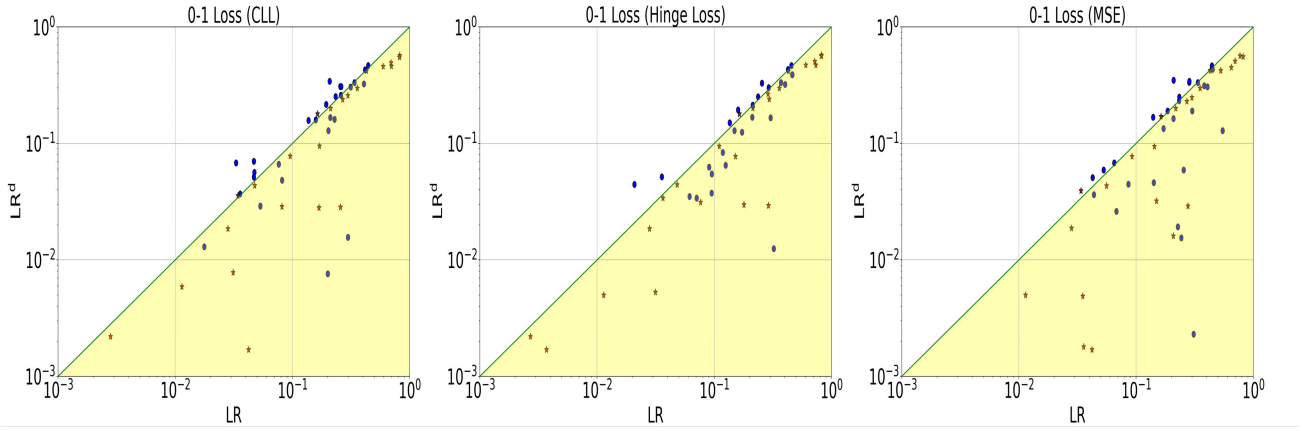
- Discretization can lead to simplification of statistical analysis. For example, if a quantitative attribute is split on the median, then one can compare the two groups based on t, χ^2 or some other test to estimate the difference between the two groups. This may ease interpretation and presentation of results [31].
- If there is error in the measurement scale, discretization can improve the performance of the model by reducing the contamination [33]–[36].
- In many domains there exist pre-defined (or standard) thresholds to convert a quantitative to a qualitative scale. In these cases, a discretized attribute might better represent the task at hand as it will be more interpretable or have distinct significance. For example, in medical research doctors might better interpret blood-pressure as high and low rather than on a numeric scale.
- A discretized attribute might be better utilized than the quantitative attribute by the learning system. For example, consider a classifier that relies on estimation of conditional probabilities such as $P(x_i \mid y)$. If X_i is quantitative, x_i can take infinite many values and if the number of training samples are small, reliable estimation of $P(x_i \mid y)$ from the data is not possible. A common approach is to impose a parametric model to estimate the value of $P(x_i \mid y)$ based on this model in which case the accuracy will depend on the appropriateness of the parametric model selected. Discretization can obviate this problem. By converting a quantitative attribute X_i into a qualitative one X_i^*, the probabilities will take the form of $P(x_i^* \mid y)$ which may be reliably estimated from the data as there will be many x_i values falling into the same interval [32].
- The final reason for discretization has to do with overcoming a model's assumptions. It might be the case that discretization help avoid some strong assumption that the learner makes about the data. If those assumptions are correct, discretization will have a negative impact, but if those assumptions are false, discretization may lead to better results [37]. It is this final motivation that we examine herein.

The effect of discretization on various classification algorithms such as naive Bayes, Support Vector Machines and Random Forest is discussed in [38] on many biomedical datasets, where it is shown that discretization can greatly improve the performance of the learning algorithm. The role of discretization as feature selection technique is also explored. On various contrived datasets, [39] studied the effect of discretization on the precision and recall of various classification methods.

The effectiveness of discretization for naive Bayes classifier is relatively well studied [4], [15], [32]. Reference [4] conducted an empirical study of naive Bayes with four well-known discretization methods and found that all the discretization methods result in significantly reducing error relative to a naive Bayes that assumes a Gaussian distribution for the continuous variables. Reference [15] attributes this to the *perfect aggregation* property of Dirichlet distributions. In naive Bayes settings, a discretized continuous distribution is assumed to have a categorical distribution with Dirichlet priors. The perfect aggregation property of Dirichlet implies that we can learn the class-conditional probability of the discretized interval with arbitrary accuracy. It is also shown that there exists a *partition independence assumption*, by virtue of that, Dirichlet parameters corresponding to a certain interval depend only on the area below the curve of the probability distribution function, but is independent of the shape of the curve in that interval.

IV. EXPERIMENTS

In this section, we compare the performance of LR with discretized LR (denoted as LR$^{\mathrm{d}}$) on various datasets from the UCI repository [40]. The details of datasets used in this



(a) LR vs. LR$^{\mathrm{d}}$ (CLL) (b) LR vs. LR$^{\mathrm{d}}$ (HL) (c) LR vs. LR$^{\mathrm{d}}$ (MSE)

FIGURE 6. [Figure is best seen in colour] Comparative scatter of 0-1 Loss results for LR and LR$^{\mathrm{d}}$ – (Left) CLL, (Middle) HL, (Right) MSE. LR is on the X-axis. LR$^{\mathrm{d}}$ is on the Y-axis. For points below the diagonal line shaded in yellow, LR$^{\mathrm{d}}$ wins. Results on *Big* datasets are shown in red stars (*), whereas results on *Little* datasets are shown in blue circles (o). Note, both axes are on log-scale.

work are given in Appendix B. For LR$^{\mathrm{d}}$, different supervised and un-supervised discretization techniques were considered (such as equal-width, equal-frequency, entropy-based, etc.). As we discussed earlier, this is not a comparative study on the relative efficacies of various discretization techniques, we only report LR$^{\mathrm{d}}$ results with supervised entropy-based discretization of [41], which we found gives better results than other discretization methods such as equal-frequency, equal-width, etc. We also considered both quasi-Newton method – L-BFGS and Truncated Newton method – Tron. The two methods provide similar results, and hence we only report Tron results in this section.

In the following, LR means a logistic regression classifier trained with original data (i.e., with both qualitative and quantitative attributes). LR$^{\mathrm{d}}$ means a logistic regression classifier trained on discretized data (i.e., quantitative attributes are converted into qualitative ones). As discussed, we will compare three variants of LR and LR$^{\mathrm{d}}$ based on different loss functions that they optimize, namely Conditional Log-Likelihood, Hinge Loss and Mean-square-error.

Each algorithm is tested on each dataset using either 5 or 10 rounds of 2-fold cross validation.

During the presentation of results, we split our datasets into two categories – *Big* and *Little*. The *Big* category comprises of datasets with more than 100, 000 instances and the *Little* category comprises of the remaining datasets with $< 100,000$ instances.

We compare four different metrics: 0-1 Loss, RMSE, Bias and Variance. We also compare training-time and rate of convergence. As discussed in Section I, the reason for performing bias-variance estimation is that it provides insights into how the learning algorithm might be expected to perform with varying amounts of data. We expect low variance algorithms to have relatively low error for small data and low bias algorithms to have relatively low error for large data [42]. There are a number of different bias-variance decomposition definitions. In this research, we use the bias and variance definitions of [43] together with the repeated cross-validation bias-variance estimation method proposed by [44].[2]

We report Win-Draw-Loss (W-D-L) results when comparing the 0-1 Loss, RMSE, bias and variance of two models. A two-tail binomial sign test is used to determine the significance of the results. Results are considered significant if $p \leq 0.05$ and shown in bold.

For hinge-loss, a dataset with more than two classes was transformed into a binary dataset. Data points belonging to the majority class were assigned to class A and the remaining data points were assigned to class B.

A. COMPARISON OF THE ACCURACY OF LR^d AND LR

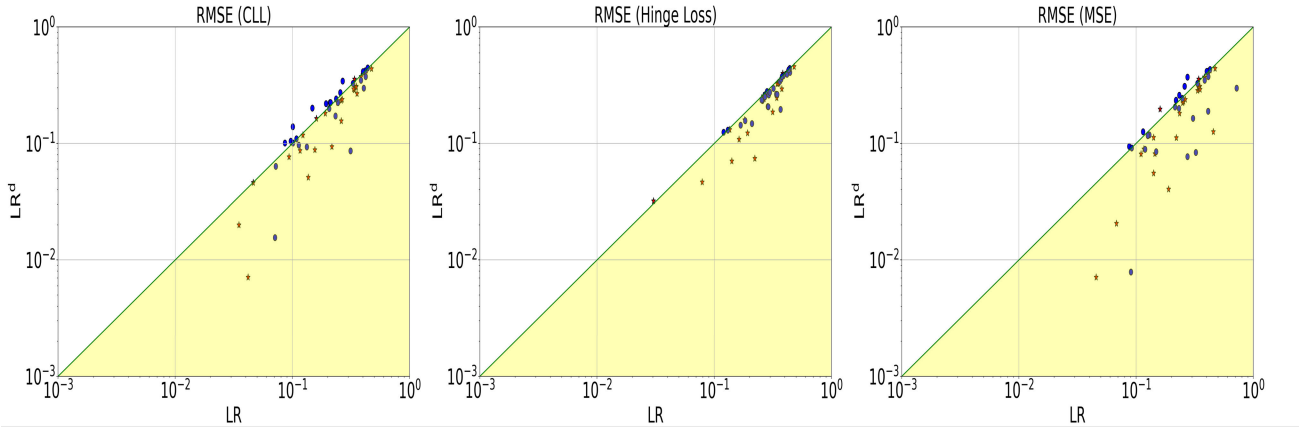
In this section, we compare the accuracy of LR and LR$^{\mathrm{d}}$ in terms of the 0-1 Loss and RMSE on 52 datasets in Figures 6 and 7. It can be seen that LR$^{\mathrm{d}}$ with all three objective functions results in much better accuracy than LR – almost all the points are below the diagonal line. The results on *Big* datasets are shown in red stars (*), whereas results on *Little* datasets are shown in blue dots (o). It can be seen that, on almost all *Big* datasets, LR$^{\mathrm{d}}$ leads to better accuracy (almost all red-stars are below the diagonal line), which is extremely encouraging. It can also be seen that some of the differences are substantial (note the axis are on the log-

[2] Reference [43] define bias and variance as follows:

$$\text{bias}^2 = \frac{1}{2}\sum_{y=1}^{C}\left(\tilde{P}(y|x) - P(y\,|\,x)\right)^2,$$

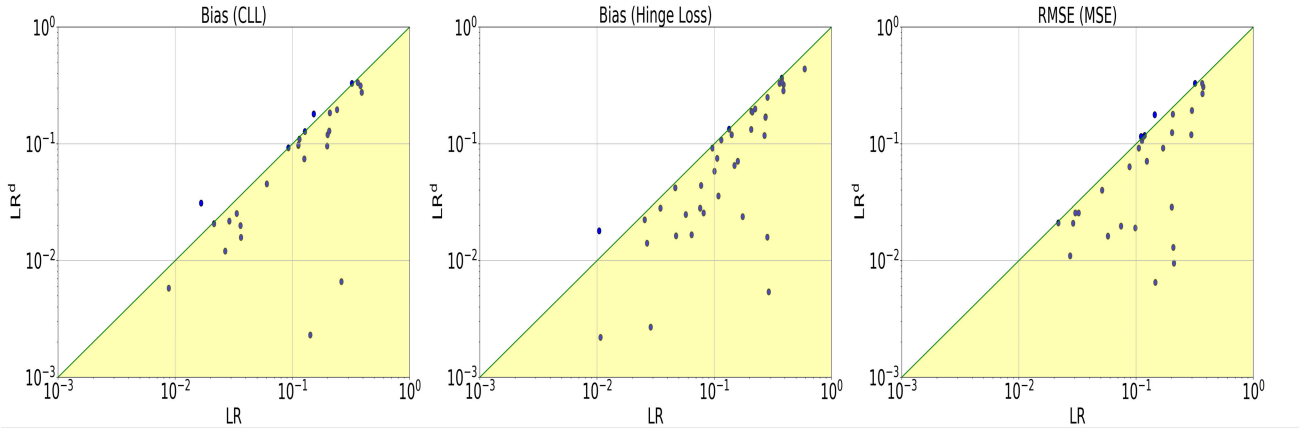
and

$$\text{variance} = \frac{1}{2}\left(1 - \sum_{y=1}^{C} P(y\,|\,x)^2\right).$$



(a) LR vs. LR^d (CLL) (b) LR vs. LR^d (CLL) (c) LR vs. LR^d (CLL)

FIGURE 7. **[Figure is best seen in colour] Comparative scatter of RMSE results for LR and LR^d – (Left) CLL, (Middle) HL, (Right) MSE. LR is on the X-axis. LR^d is on the Y-axis. For points below the diagonal line shaded in yellow, LR^d wins. Results on *Big* datasets are shown in red stars (*), whereas results on *Little* datasets are shown in blue circles (o). Note, both axes are on log-scale.**



(a) LR vs. LR^d (CLL) (b) LR vs. LR^d (CLL) (c) LR vs. LR^d (CLL)

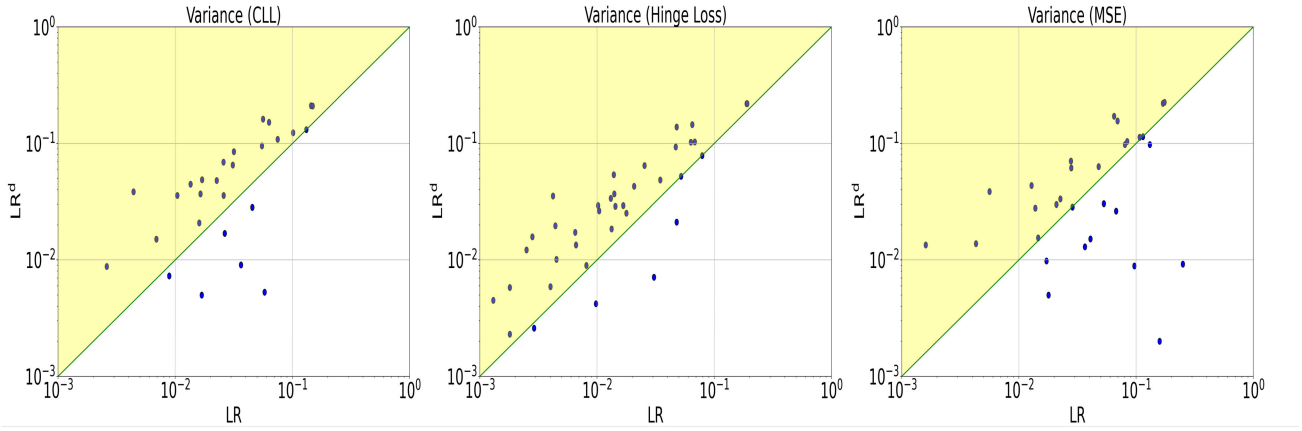
FIGURE 8. **[Figure is best seen in colour] Comparative scatter of Bias results for LR and LR^d – (Left) CLL, (Middle) HL, (Right) MSE. LR is on the X-axis. LR^d is on the Y-axis. For points below the diagonal line shaded in yellow, LR^d wins. Note, both axes are on log-scale.**

scale) – this shows the effectiveness of discretization on LR especially for big datasets.

A comparison of the win-draw-loss between the two models is given in Table 2, which provides the similar information as shown in Figures 6 and 7, albeit in different way. It can be seen that on big datasets, LR^d wins on all except on 2 datasets – very promising result. This proves our hypothesis that on big datasets, discretization leads to low-bias non-linear classifier resulting in far superior results than LR with no discretization. On small datasets, discretization is significantly effective for Hinge-loss and non-significantly effective for MSE. With CLL (on small datasets), LR^d and LR leads to similar performances with 13 wins and 14 losses for 0-1 Loss and 12 wins and 15 losses for RMSE. However, one should take into account that the scale of LR^d wins is much higher than that of LR. This can be seen from the spread of blue-dots in the left-most plots of Figures 6 and 7.

TABLE 2. **Win-Draw-Loss comparison of 0-1 Loss and RMSE of LR^d vs. LR with CLL, MSE and HL. Significant results are shown in bold.**

	LR^d vs. LR CLL		**LR^d vs. LR** MSE		**LR^d vs. LR** HL	
	W-D-L	p	W-D-L	p	W-D-L	p
			All Datasets			
0-1 Loss	35/1/16	**0.011**	39/1/12	**< 0.001**	41/1/10	**< 0.001**
RMSE	34/1/7	**0.016**	39/1/12	**< 0.001**	47/1/4	**< 0.001**
			Big Datasets			
0-1 Loss	22/0/2	**< 0.001**	22/0/2	**< 0.001**	23/0/1	**< 0.001**
RMSE	22/0/2	**< 0.001**	22/0/2	**< 0.001**	22/0/2	**< 0.001**
			Small Datasets			
0-1 Loss	13/1/14	1.000	17/1/10	0.247	18/1/9	0.087
RMSE	12/1/15	0.701	17/1/10	0.247	25/1/2	**< 0.001**



(a) LR vs. LR^d (CLL) (b) LR vs. LR^d (CLL) (c) LR vs. LR^d (CLL)

FIGURE 9. [Figure is best seen in colour] Comparative scatter of Variance results for LR and LR^d – (Left) CLL, (Middle) HL, (Right) MSE. LR is on the X-axis. LR^d is on the Y-axis. For points above the diagonal line shaded in yellow, LR^d loses. Note, both axes are on log-scale.

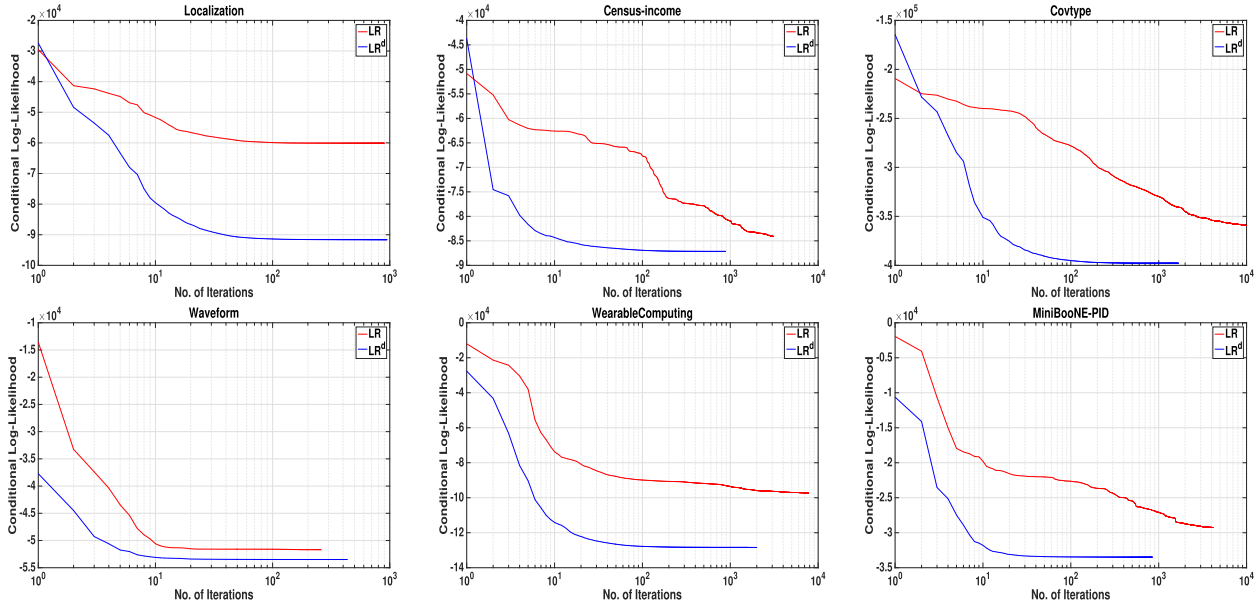


FIGURE 10. [Figure is best seen in colour] Comparison of the rate of convergence of LR and LR^d with NLL on six sample datasets. The X-axis is on log scale.

B. BIAS-VARIANCE ANALYSIS OF LR^d AND LR

Let us compare the bias and variance of LR and LR^d. We plot the scatter plots of the bias of LR and LR^d with three objective functions in Figure 8 and the scatter plots of the variance in Figure 9. From the scatter plots, it can be seen that LR^d leads to a low-bias and high-variance model. One can see that majority of points lies below the diagonal line in Figure 8 – demonstrating that LR^d is lower biased than LR. Whereas, majority of points lies above the diagonal line in Figure 9 – demonstrating that LR^d has much higher variance than LR.

Note that we present a bias-variance analysis on only *Little* category of datasets. The reason is that results were obtained in a heterogeneous environment (cluster computing). Of course, to compute the bias and variance, one needs to run the algorithms multiple time resulting in many iterations. For speeding-up the algorithm, one can actually run these iterations in parallel on different computers in the cluster. The downside of this (many-fold) speed-up is that one can not compute the bias and variance without managing the inter-node communication. Therefore, we computed the bias and variance results on a desktop computer with limited memory. Nonetheless, results confirms our hypothesis of LR^d being low-biased as it has more parameters than LR. This low-bias translated into better performance on *Big* datasets as discussed in the previous section.

C. COMPARISON OF THE CONVERGENCE CURVES OF LR^d AND LR

As both LR^d and LR are trained via iterative optimization algorithms, they produce a sequence of values during the optimization process – i.e., of their objective function

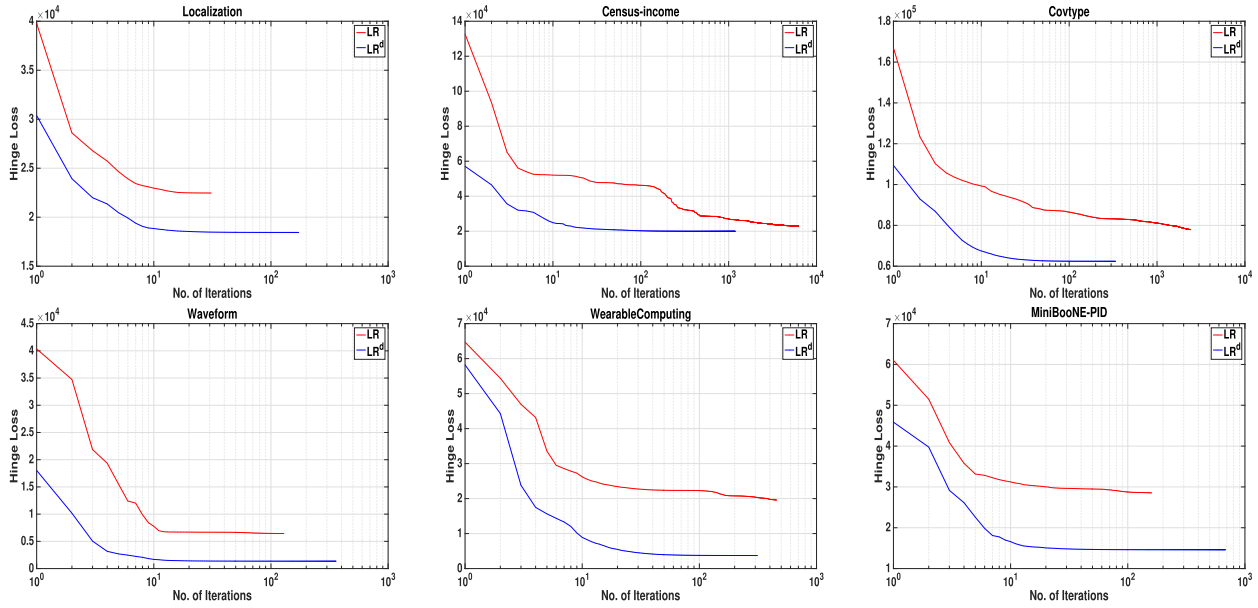


FIGURE 11. [Figure is best seen in colour] Comparison of the rate of convergence of LR and LR^{d} with Hinge Loss on six sample datasets. The X-axis is on the log scale.

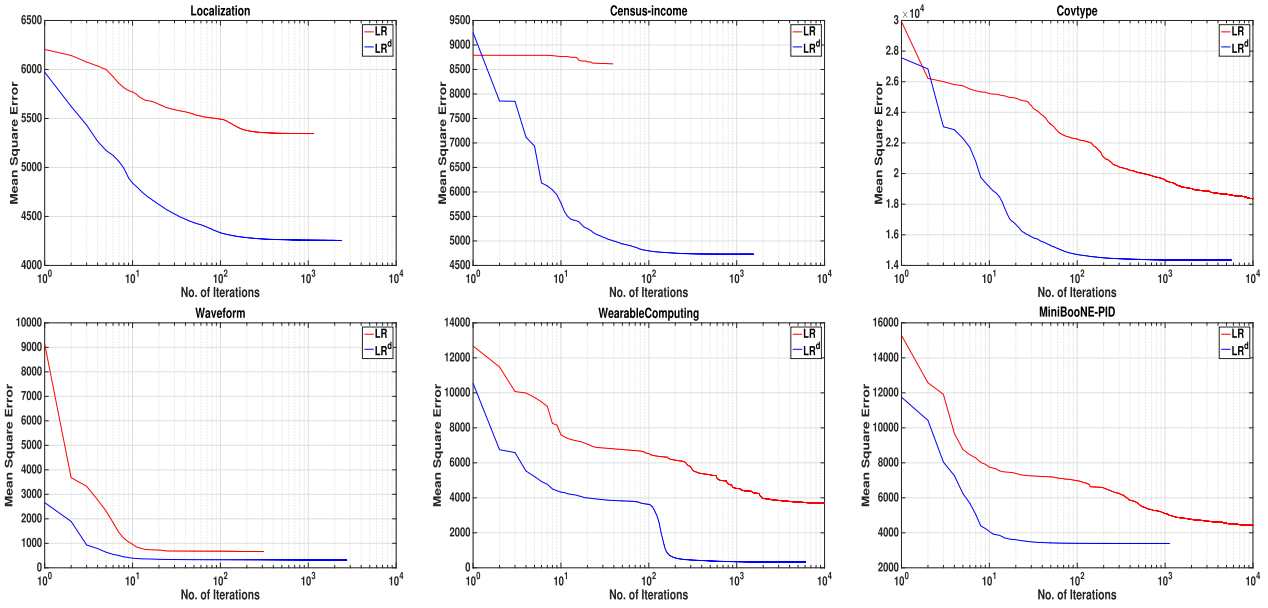
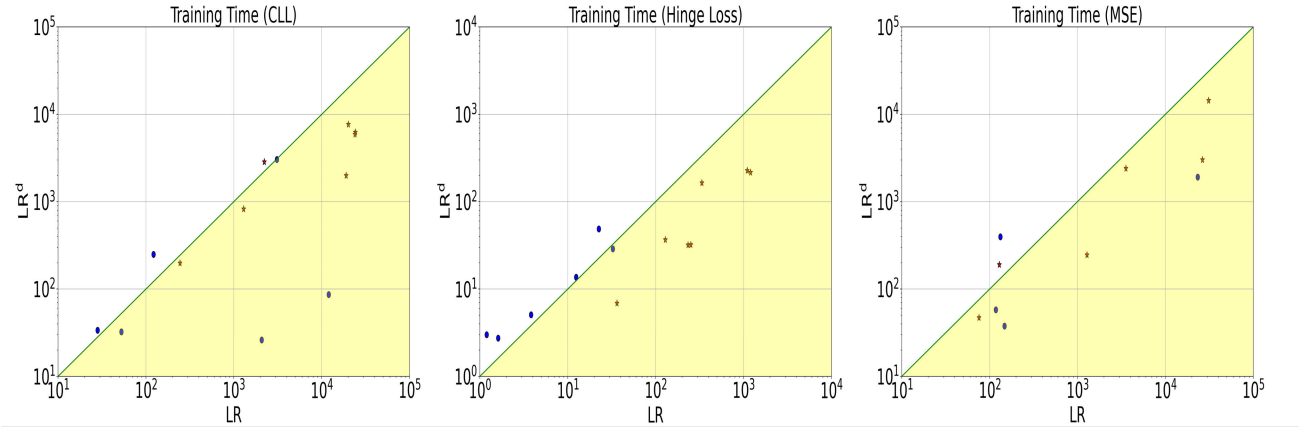


FIGURE 12. [Figure is best seen in colour] Comparison of rate of convergence of LR and LR^{d} with Mean Square Error on six sample datasets. The X-axis is on the log scale.

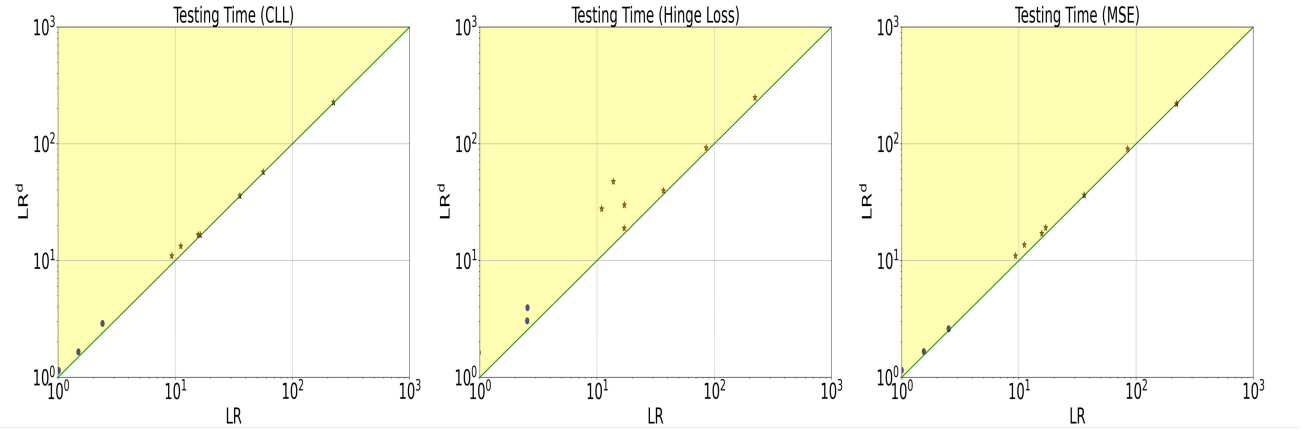
which (should ideally) decrease with successive iterations until convergence. A technique that leads to the global minimum faster (steeper curve) and in fewer iterations (shorter curve) is desirable. Note that both LR and LR^{d} are different models (due to their parameterizations) and, therefore, the optimization space for the two problems is also very different. We leave it to future work, to compare the optimization space of the two models. In the following, let us compare the convergences of LR and LR^{d} on some sample datasets. A similar trend was observed on all datasets, here we report results on six representative datasets only, due to space constraints.

A comparison of the variation in negative of conditional-log-likelihood (NLL) objective function for LR and LR^{d} is shown in Figure 10. It can be seen that LR^{d} has steeper curve – that is, it asymptotes to its global minimum much quickly. It is also important to see that LR^{d} leads to much lower NLL. Better accuracy of LR^{d} is the result of this much lower NLL. Figures 11 and 12 shows the variation in HL and MSE, respectively. A similar trend to NLL can be see, where



(a) LR vs. LR$^{\mathrm{d}}$ (CLL) (b) LR vs. LR$^{\mathrm{d}}$ (HL) (c) LR vs. LR$^{\mathrm{d}}$ (MSE)

FIGURE 13. [Figure is best seen in colour] Comparative scatter of Training-time results for LR and LR$^{\mathrm{d}}$ – (Left) CLL, (Middle) HL, (Right) MSE. LR is on the X-axis. LR$^{\mathrm{d}}$ is on the Y-axis. For points below the diagonal line shaded in yellow, LR$^{\mathrm{d}}$ wins. Results on *Big* datasets are shown in red stars (*), whereas results on *Little* datasets are shown in blue circles (o). Note, both axes are on log-scale.



(a) LR vs. LR$^{\mathrm{d}}$ (CLL) (b) LR vs. LR$^{\mathrm{d}}$ (CLL) (c) LR vs. LR$^{\mathrm{d}}$ (CLL)

FIGURE 14. [Figure is best seen in colour] Comparative scatter of Classification-time results for LR and LR$^{\mathrm{d}}$ – (Left) CLL, (Middle) HL, (Right) MSE. LR is on the X-axis. LR$^{\mathrm{d}}$ is on the Y-axis. For points above the diagonal line shaded in yellow, LR$^{\mathrm{d}}$ loses. Results on *Big* datasets are shown in red stars (*), whereas results on *Little* datasets are shown in blue circles (o). Note, both axes are on log-scale.

LR$^{\mathrm{d}}$ leads to a better value of the objective function while converging more rapidly.

We argue that that this is extremely important and significant result. Not only discretization has led to a non-linear model, but in the transformed space where features can only take binary values, is somehow more easy for optimization solver to traverse and find the global minima in a much speedy manner.

D. COMPARISON OF THE LEARNING AND CLASSIFICATION TIME LR^d AND LR

In this section, we compare the training and classification time of LR and LR$^{\mathrm{d}}$. It can be seen from Figure 13 that LR$^{\mathrm{d}}$ trained with CLL and MSE is slightly faster than LR (majority of points below the diagonal line), whereas LR and LR$^{\mathrm{d}}$ with hinge loss leads to a similar training-time profile. We already have seen the superior classification performance of LR$^{\mathrm{d}}$. These training-time results are extremely encouraging as they suggest that LR$^{\mathrm{d}}$ can result in much better classification accuracy without compromising the computational performance. We have reported the results only on a subset of *Little* and *Big* datasets. This is because, the results were obtained by running the jobs on the local-desktop computer (i.e., a controlled set-up), rather than in a cluster-computing environment. The scatter plots of classification time results for LR and LR$^{\mathrm{d}}$ with three objective functions are presented in Figure 14. It can be seen that LR has slightly better classification time than LR$^{\mathrm{d}}$.

V. CONCLUSION AND FUTURE WORKS

In this article, we studied the role of discretization for linear classifiers in machine learning. We discussed the typical use-case of discretization in typical machine learning models, i.e., when the model requires only qualitative attributes.

TABLE 3. Details of datasets used in this article.

Domain	Case	Qualitative	Quantitative	Classes	Missing Values	Source
HIGGS	11000000	0	28	2	N	UCI
HEPMASS	10500000	0	27	2	N	UCI
kddcup	5209460	7	34	40	N	UCI
SUSY	5000000	0	18	2	N	UCI
Watch_accelerometer	3540962	0	3	7	N	UCI
Watch_gyroscope	3205431	0	3	7	N	UCI
Phone_gyroscope(40%)	2786526	0	3	7	N	UCI
Phone_accelerometer(40%)	2612495	0	3	7	N	UCI
satellites(25%)	2176290	0	138	24	Y	UCI
PAMAP2(25%)	962626	1	53	19	N	UCI
MITFaceSetC	839330	0	361	2	N	Non-UCI (Libsvm)
covertype	581012	44	10	7	N	UCI
MITFaceSetB	489410	0	361	2	N	Non-UCI (Libsvm)
MITFaceSetA	474101	0	361	2	N	Non-UCI (Libsvm)
USPESExtended	341462	0	675	2	N	Non-UCI (Libsvm)
census-income(KDD)	299285	32	9	2	N	UCI
SkinSegmentation	245057	0	3	2	N	UCI
WearableComputing	165632	2	15	5	N	UCI
localization	164860	2	3	11	N	UCI
TwitterAbsoluteSigma500	140607	0	76	2	N	UCI
MiniBooNE_PID	130065	0	50	2	N	UCI
TVNewsChannelCommercial	129685	0	4124	2	N	UCI
Diabetes	101766	52	9	3	Y	UCI
waveform	100000	0	20	3	N	UCI
shuttle	58000	0	9	7	N	UCI
adult	48842	8	6	2	N	UCI
letter-recog	20000	0	16	24	N	UCI
magic	19020	0	10	2	N	UCI
sign	12546	0	8	3	N	UCI
pendigits	10992	0	16	10	N	UCI
pioneer	9150	7	29	57	N	UCI
satellite	6435	0	35	6	N	UCI
optdigits	5620	0	64	10	N	UCI
page-blocks	5473	0	10	5	N	UCI
wall-following	5456	0	24	4	N	UCI
phoneme	5438	0	7	50	N	UCI
waveform-5000	5000	0	40	3	N	UCI
spambase	4601	0	57	2	N	UCI
abalone	4177	0	8	3	N	UCI
segment	2310	0	19	7	N	UCI
mfeat-mor	2000	2	3	10	N	UCI
volcanoes	1520	0	3	4	N	UCI
yeast	1484	1	7	10	N	UCI
vowel	990	3	10	11	N	UCI
vowel-context	990	1	10	11	N	UCI
vehicle	946	0	18	4	N	UCI
anneal	798	32	6	3	N	UCI
pid	768	0	8	2	N	UCI
syncon	600	0	60	6	N	UCI
musk1	476	0	166	2	N	UCI
new-thyroid	215	0	5	3	N	UCI
wine	178	0	13	3	N	UCI

We discussed that there exists significant aversion towards discretization as it loses information. We argued that the role of discretization is more than a mere pre-processing technique. We argued that discretization – in spite of losing information, can help model non-linear relationships in the data and, therefore, can help reduce the representation bias of a learner that uses linear models. A linear classifier trained on discretized data is not linear any more which has the potential to help in modelling non-linear decision boundaries which otherwise, would require the use of non-linear models such as decision trees, multi-layer networks, etc.

We showed that discretization can greatly help improve the performance of logistic regression and other linear classifiers optimizing Hinge Loss and Mean-square-error especially on large datasets. We compared the performance of LR trained with both qualitative and quantitative attributes with LR trained with qualitative attributes only, where quantitative attributes were discretized first. Our empirical analysis on

52 datasets showed that LR with discretization led to a low-bias model and, therefore, it resulted in significantly better 0-1 Loss and RMSE performance on large datasets. Quite surprisingly, it also reduced the training time and had more desirable convergence.

With faster training, better convergence and low-bias we believe that discretization is worth consideration in any context where linear classifiers are learned from quantitative data.

There are several research questions that have arisen out of this research, that we would like to pursue:

- Given the effectiveness of discretization for linear models, it is worth analysing discretization for non-linear models as well. This is motivated from faster convergence behaviour of models trained on discretized data. We conjecture that the optimization space span by discretized data can be useful for even non-linear models such as deep Artificial Neural Networks, which notoriously suffer greatly due to the existence of saddle points.
- We are interested to explore the properties of the optimization space span by discretized data and why it leads to better convergence profile than the original data.
- In our preliminary results, use of first-order optimization methods such as Adam and Adagrad, led to similar results to that of Tron and L-BFGS. However, a systematic study of these optimization methods is needed.
- A systematic analysis of other loss functions would be beneficial as well, to demonstrate that the trend is general across other loss functions.
- Investigating embeddings for quantitative features produced as a result of discretization has the potential to produce much better results than the use of 1-hot-encoding that is currently used in this work. We see this as an exciting new direction of this work.

VI. CODE

The details of the software library `fastLC` is given in Appendix A. The library along with running instructions can be downloaded from Github: https://github.com/nayyarzaidi/fastLC.git.

APPENDIX A
`fastLC`-LR LIBRARY

The library can handle both quantitative and qualitative attributes. There is no need to do a one-hot-encoding for qualitative attributes, as the LR model built can actually handle the data types.

One can execute the code in the library by issuing the following command for LR with CLL: ≫ `java -cp /fastLC.jar fastLC.BVDcrossvalx-t/dataset.arff -i 2 -x 2 -W LR.LRClassifier --V -S "overparamLR" -O "Tron"`.

For LR with HL, use: ≫ `java -cp /fastLC.jar fastLC.BVDcrossvalx -t /dataset. arff -i 2 -x 2 -W SVC.SVCClassifier - -V -S"overparamSVC" -O "Tron"`.

For LR with MSE, use following command: ≫ `-cp /fastLC.jar fastLC.BVDcrossvalx -t /dataset.arff -i 2 -x 2 -W ANN.ANNClassifier - -V -S "overparamANN" -O "Tron"`,

Note, `-i 2 -x 2` specifies how many rounds of how many iterations, `-V` is the verbosity flag, whereas, `-O` specifies the solver. One can choose from the following list: {GD, QN, CG, Tron, SGD} for gradient descent, conjugate gradient, truncated Newton and stochastic gradient descent.

The default implementations learns parameters for all (C) classes. One can force the library to run weights only for $C-1$ classes. This can be done by calling `-S "vanillaLR"`, `-S "vanillaSVC"` and `-S "vanillaMSE"` for LR with CLL, HL and MSE objective functions respectively.

For computing results on discretized data, either pre discretize the dataset, or use the `-D` flag to convert quantitative attributes into qualitative one by the learner.

APPENDIX B
DETAILS OF DATASETS

Details of datasets used in this article are shown in Table 3.

REFERENCES

[1] N. A. Zaidi, M. J. Carman, J. Cerquides, and G. I. Webb, "Naive-Bayes inspired effective pre-conditioner for speeding-up logistic regression," in *Proc. IEEE Int. Conf. Data Mining*, Dec. 2014, pp. 1097–1102.
[2] P. van der Putten and M. van Someren, "A bias-variance analysis of a real world learning problem: The coil challenge 2000," *Mach. Learn.*, vol. 57, no. 1, pp. 177–195, 2004.
[3] T. M. Mitchell, "The need for biases in learning generalizations," Dept. Comput. Sci., Rutgers Univ., Camden, NJ, USA, Tech. Rep. CBM-TR-117, 1980.
[4] J. Dougherty, R. Kohavi, and M. Sahami, "Supervised and unsupervised discretization of continuous features," in *Proc. ICML*, 1995.
[5] S. Garcia, J. Luengo, J. A. Sáez, V. López, and F. Herrera, "A survey of discretization techniques: Taxonomy and empirical analysis in supervised learning," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 4, pp. 734–750, Apr. 2013.
[6] N. A. Zaidi, F. Petitjean, and G. I. Webb, "Efficient and effective accelerated hierarchical higher-order logistic regression for large data quantities," in *Proc. SIAM SDM*, 2018, pp. 459–467.
[7] N. A. Zaidi, G. I. Webb, M. J. Carman, F. Petitjean, W. Buntine, M. Hynes, and H. De Sterck, "Efficient parameter learning of Bayesian network classifiers," *Mach. Learn.*, vol. 106, nos. 9–10, pp. 1289–1329, Oct. 2017.
[8] B. D. Ripley, *Pattern Recognition and Neural Networks*. Cambridge, U.K.: Cambridge Univ. Press, 1996.
[9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org
[10] L. Rosasco, E. De Vito, A. Caponnetto, M. Piana, and A. Verri, "Are all loss functions the same?" *Neural Comput.*, vol. 16, pp. 1–35, Sep. 2004.
[11] I. H. Witten, E. Frank, and M. A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd ed. San Mateo, CA, USA: Morgan Kaufmann, 2011.
[12] N. A. Zaidi, F. Petitjean, and G. I. Webb, "Preconditioning an artificial neural network using Naive Bayes," in *Proc. Adv. Knowl. Discovery Data Mining*, 2016, pp. 341–353.
[13] H. Liu, F. Hussain, C. L. Tan, and M. Dash, "Discretization: An enabling technique," *Mach. Learn.*, vol. 6, no. 4, pp. 393–423, 2002.
[14] R. Kohavi and M. Sahami, "Error-based and entropy-based discretization of continuous features," in *Proc. AAAI*, 1996, pp. 114–119.
[15] C. N. Hsu, H. J. Huang, and T. T. Wong, "Why discretization works for Naive Bayesian classifiers," in *Proc. Int. Conf. Mach. Learn.*, 2000, pp. 309–406.
[16] C. N. Hsu, H. J. Huang, and T. T. Wong, "Implications of the Dirichlet assumption for discretization of continuous variables in Naive Bayesian classifiers," *Mach. Learn.*, vol. 53, no. 3, pp. 235–263, 2003.

[17] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 3111–3119.
[18] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proc. EMNLP*, vol. 14, 2014, pp. 1532–1543.
[19] C. Guo and F. Berkhahn, "Entity embeddings of categorical variables," 2016, *arXiv:1604.06737*. [Online]. Available: http://arxiv.org/abs/1604.06737
[20] Y. Qu, B. Fang, W. Zhang, R. Tang, M. Niu, H. Guo, Y. Yu, and X. He, "Product-based neural networks for user response prediction over multi-field categorical data," *ACM Trans. Inf. Syst.*, vol. 37, no. 1, 2018.
[21] Y. Wen, T. Chen, J. Wang, and W. Zhang, "Pairwise multi-layer nets for learning distributed representation of multi-field categorical data," in *Proc. 1st Int. Workshop Deep Learn. Pract. High-Dimensional Sparse Data*, 2019, pp. 1–8.
[22] J. Nocedal and S. Wright, *Numerical Optimization*. Springer, 2006.
[23] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, no. 7, pp. 2121–2159, 2011.
[24] T. Tieleman and G. Hinton, "Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude," in *Proc. Neural Netw. Mach. Learn. (COURSERA)*, 2012.
[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*. [Online]. Available: http://arxiv.org/abs/1412.6980
[26] S. G. Nash, "A survey of truncated Newton methods," *J. Comput. Appl. Math.*, vol. 124, nos. 1–2, pp. 45–59, 2000.
[27] N. A. Zaidi and G. I. Webb, "A fast trust-region Newton method for softmax logistic regression," in *Proc. SIAM SDM*, 2017, pp. 705–713.
[28] J. R. Irwin and G. H. McClelland, "Negative consequences of dichotomizing continuous predictor variables," *J. Marketing Res.*, vol. 40, no. 3, pp. 366–371, Aug. 2003.
[29] R. C. MacCallum, S. Zhang, K. J. Preacher, and D. D. Rucker, "On the practice of dichotomization of quantitative variables," *Psychol. Methods*, vol. 7, no. 1, pp. 10–40, 2002.
[30] S. Greenland, "Dose-response and trend analysis in epidemiology: Alternatives to categorical analysis," *Epidemiology*, vol. 6, no. 4, pp. 356–365, Jul. 1995.
[31] D. G. Altman and P. Royston, "The cost of dichotomising continuous variables," *Brit. Med. J.*, vol. 332, no. 7549, May 2006, Art. no. 1080.
[32] Y. Yang and G. I. Webb, "Discretization for naive-bayes learning: Managing discretization bias and variance," *Mach. Learn.*, vol. 74, no. 1, pp. 39–74, Jan. 2009.
[33] C. Flegal, P. Keyl, and P. Nieto, "Differential misclassification arising from nondifferential errors in exposure measurement," *Amer. J. Epidemiol.*, vol. 34, no. 10, pp. 1233–1244, 1991.
[34] S. Reade-Christopher and L. Kupper, "Effect of exposure misclassification on regression analysis," *Biometrics*, vol. 47, pp. 535–548, Jan. 1991.
[35] K. Y. Fung and G. R. Howe, "Methodological issues in case-control studies. III: The effect of joint misclassification of risk factors and confounding factors upon estimation and power," *Int. J. Epidemiol.*, vol. 13, pp. 366–370, 1984.
[36] Y. Shentu and M. Xie, "A note on dichotomization of continuous response variable in the presence of contamination and model misspecification," *Statist. Med.*, vol. 29, no. 21, pp. 2200–2214, Sep. 2010.
[37] D. Altman, B. Lausen, W. Sauerbrei, and M. Schumacher, "Dangers of using 'optimal' cutpoints in the evaluation of prognostic factors," *J. Nat. Cancer Inst.*, vol. 86, no. 11, pp. 829–835, 1994.
[38] J. L. Lustgarten, V. Gopalakrishnan, H. Grover, and S. Visweswaran, "Improving classification performance with discretization on biomedical datasets," in *Proc. AMIA Symp.*, 2008, p. 445.
[39] R. E. Maleki, S. M. Iranmanesh, and B. M. Bidgoli, "An experimental investigation of the effect of discrete attributes on the precision of classification methods," in *Proc. Inf. Commun. Technol.*, 2009, pp. 215–220.
[40] M. Lichman. (2017). *UCI Machine Learning Repository*. [Online]. Available: http://archive.ics.uci.edu/ml
[41] U. M. Fayyad and K. B. Irani, "On the handling of continuous-valued attributes in decision tree generation," *Mach. Learn.*, vol. 8, no. 1, pp. 87–102, Jan. 1992.
[42] D. Brain and G. I. Webb, "The need for low bias algorithms in classification learning from large data sets," in *Proc. PKDD*, 2002, pp. 62–73.
[43] R. Kohavi and D. Wolpert, "Bias plus variance decomposition for zero-one loss functions," in *Proc. ICML*, 1996, pp. 275–283.
[44] G. I. Webb, "Multiboosting: A technique for combining boosting and wagging," *Mach. Learn.*, vol. 40, no. 2, pp. 159–196, 2000.

NAYYAR A. ZAIDI received the B.S. degree in computer science and engineering from the University of Engineering and Technology, Lahore, Pakistan, in 2005, and the Ph.D. degree in artificial intelligence from Monash University, Melbourne, VIC, Australia, in 2011.

He worked as a Research Fellow, a Lecturer, and a Research Fellow, from 2011 to 2013, from 2013 to 2014, and from 2014 to 2017, respectively, at the Faculty of Information Technology, Monash University. From 2017 to 2019, he worked as Research Scientist at Credit AI (Trusting Social) Melbourne Lab. Since 2020, he has been working as a Senior Lecturer of Computer Science at Deakin University, Melbourne. His research interests include effective feature engineering, explainable model, uncertainty prediction, and reinforcement learning. He is also interested in practical data science, machine learning engineering, and data science trainings. He was a recipient of the Gold Medal for graduating top of the class at the University of Engineering and Technology.

YANG DU received the master's degree in information technology from Monash University. He is currently a Software Engineer, developing extensions and plugins for various e-commerce platforms. He was working on a project exploring how discretization methods affect the output from logistic regression classification model. He is also interested in the recommender systems and data visualization.

GEOFFREY I. WEBB (Fellow, IEEE) is currently the Research Director of the Monash Data Futures Institute, Monash University. He is a Technical Advisor to BigML, Inc., who have incorporated his best of class association discovery software, Magnum Opus, as a core component of their cloud-based machine learning service. He has developed many of the key mechanisms of support-confidence association discovery in the 1980s. His OPUS search algorithm remains the state-of-the-art in rule search. He pioneered multiple research areas as diverse as black-box user modeling, interactive data analytics, and statistically sound pattern discovery. He has developed many useful machine learning algorithms that are widely deployed. He has published more than 200 scientific articles. He is the author of the Magnum Opus commercial data mining software package, a system that embodies many of his research contributions in the area of data mining and has contributed many components to the popular Weka machine learning workbench. He is a leading Data Scientist and the only Australian to have been the Program Committee Chair of the two leading data mining conferences, ACM SIGKDD and IEEE ICDM.

Prof. Webb's numerous awards include the inaugural Eureka Prize for Excellence in Data Science, in 2017. He was the Editor-in-Chief of *Data Mining and Knowledge Discovery*, from 2005 to 2014. He has been the Chief Investigator on competitive grants totalling more than $24 million.