Accepted for publication in *Machine Learning*.

ALR^n: Accelerated Higher-Order Logistic Regression

Nayyar A. Zaidi · Geoffrey I. Webb · Mark J. Carman · François Petitjean · Jesús Cerquides

Received: date / Accepted: date

Abstract This paper introduces *Accelerated Logistic Regression*: a hybrid generative-discriminative approach to training Logistic Regression with high-order features. We present two main results: (1) that our combined generative-discriminative approach significantly improves the efficiency of Logistic Regression and (2) that incorporating higher order features (i.e. features that are the Cartesian products of the original features) reduces the bias of Logistic Regression, which in turn significantly reduces its error on large datasets. We assess the efficacy of Accelerated Logistic Regression by conducting an extensive set of experiments on 75 standard datasets. We demonstrate its competitiveness, particularly on large datasets, by comparing against state-of-the-art classifiers including Random Forest and Averaged n-Dependence Estimators.

1 Introduction

Machine learning is confronted with ever growing data quantity [9]. However, many state-of-the-art learning algorithms were developed in the context of relatively small datasets. Large training sets often support the creation of very detailed models that can encode complex high-order multi-variate distributions, whereas such models would over-fit small training datasets and should be avoided [3]. We highlight this phenomenon in Figure 1. We know that the accuracy of most classifiers increases as they are provided with more training data. This can be observed in Figure 1 which plots the variation in error-rate of two classifiers with increasing quantities of training data on the `poker-hand`

Nayyar A. Zaidi, Geoffrey I. Webb, Mark J. Carman, François Petitjean
Faculty of Information Technology, Monash University, Clayton, VIC 3800, Australia
E-mail: {nayyar.zaidi, geoff.webb, mark.carman, francois.petitjean}@monash.edu

Jesús Cerquides
IIIA-CSIC, Artificial Intelligence Research Institute, Spanish National Research Council, Campus UAB, 08193 Bellaterra, Spain
E-mail: cerquide@iiia.csic.es

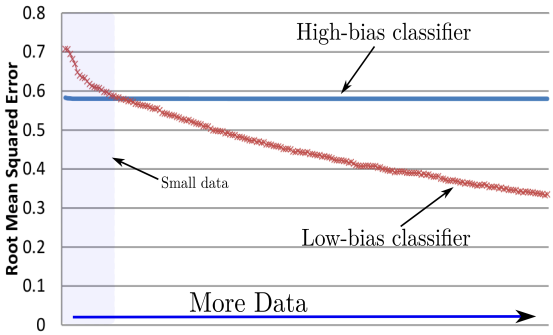

Fig. 1: Comparative study of the error committed by high- and low-bias classifiers on increasing quantities of data.

dataset [8]. One is a low-bias and high-variance learner (KDB $k = 5$, a K-Dependence Bayes estimator, taking into account quintic features, [28]) and the other is a low-variance and high-bias learner (naive Bayes, a linear classifier). For small quantities of data, the low-variance learner achieves the lowest error. However, as the data quantity increases, the low-bias learner comes to achieve the lower error as it can better model higher-order distributions from which the data might be sampled.

It has been shown that Bayesian Network Classifiers (BNCs) that explicitly represent higher-order interactions tend to have lower bias than those that do not [21,37]. This is because BNCs that can represent higher-order interactions can exactly represent any of a superset of the distributions that can be represented by BNCs that are restricted to lower order interactions. Thus they have lower *representation bias* and hence, all other things being equal, lower *inductive bias* [22] than the more restricted BNCs. Except in the specific cases where the true distribution to be modeled fits exactly into the more restricted model, given sufficient data the more expressive BNC will form a more accurate model.

It has also been shown that Logistic Regression (LR) tends to have lower bias than naive Bayes, which is a Bayesian Network Classifier with a model of equivalent form to that of LR [39,40].[1] In consequence, it seems likely that variants of LR that explicitly represent higher-order interactions should have low bias as well, and that the bias should continue to decrease as the order of the interactions represented increases. We call such variants of LR – *Higher-Order LR* – and abbreviate them as LR^n, where n is the order of interactions that are modeled. Formal definitions of these concepts are provided in Section 5.

[1] Naive Bayes and LR are generally categorized as generative and discriminative counterparts of each other. The number of the parameters of the two models are exactly the same. They only differ in the way the parameters are learned. For Naive Bayes, the parameters are actual probabilities and are learned by maximizing log-likelihood of the data and for LR, they are free parameters that are learned by optimizing the conditional log-likelihood.

While the use of higher-order LR models is quite common and at least one implementation of LR^2 is in the public domain [16], its performance relative to standard LR as data quantities vary, investigations of LR^3 and bias/variance profile of higher-order LR models warrants further investigation. We investigate these issues herein. It is noteworthy, that a significant amount of research has been done on correcting the *estimation* bias of Logistic Regression [7,34]. Most of this research has been driven by the fact that LR's parameters are obtained through Maximum Likelihood Estimation (MLE) which can be biased if data sample size is too small. However, it is shown that, asymptotically, MLE estimates will have zero *estimation* bias. Similarly, several studies have addressed the issue of bias due to omitted covariates in Logistic Regression models [23,13]. Some studies have also investigated the Bayesian version of Logistic Regression [10].

An LR^n model must be learned discriminatively through computationally intensive gradient-descent-based search. Considering all possible higher-order features in LR^n and learning the corresponding parameter by optimizing conditional log-likelihood (CLL) is a computationally intensive task. Any speed-up to the optimization process is highly desirable. A second objective of this paper is to provide an effective mechanism for achieving this.

It has been shown that a hybrid generative-discriminate learner can exploit the strengths of both Naive Bayes (NB) and Logistic Regression (LR) classifiers by creating a weighted variant of NB in which the weights are optimized using a discriminative objective function, that is, maximization of conditional log-likelihood [40,39]. The resulting model can be viewed as either using weights to alleviate the feature independence assumption of NB, or as using the maximum likelihood parameterization of NB to pre-condition the discriminative search of LR. The result is a learner that learns models that are exactly equivalent to LR, but does so much more efficiently. In this work, we show how to achieve the same result with LR^n.

We create a hybrid generative-discriminative learner named ALR^n for categorical data that learn models of equivalent order to those of LR^n, but does so much more efficiently than LR^n. We further demonstrate that the resulting models have low bias, which leads to very low error on large quantities of data. However, in order to create this hybrid learner we must first create an efficient generative counterpart to LR^n.
In summary, the contributions of this work are:

- developing an efficient generative counter-part to LR^n, named Averaged n-Join Estimators (AnJE);
- developing ALR^n: a hybrid of LR^n and AnJE;
- demonstrating that ALR^n has equivalent error to LR^n, but is substantially more efficient,
- demonstrating that ALR^n has low error on large data.

Note that it was initially proposed in [3] that for larger quantities of data, one should aim for low-bias models. This hypothesis was tested in the context of Bayesian Network classifiers in [37,36] where the results corroborated the

hypothesis. However, we are not aware of any past work that investigates this hypothesis in the context of higher-order Logistic Regression. Therefore, another contribution of this paper is:

– demonstrating that the bias of LR^n decreases as n increases and that in consequence LR^n with higher n tends to achieve lower error with greater data quantities.

The rest of this paper is organized as follows. In Section 2, we introduce the notation that is used through-out this paper. We introduce higher-order Logistic Regression in Section 3. We evaluate LR^n empirically, and show that higher values of n lead to lower-bias. Using generative models to pre-condition discriminative learning is discussed in Section 4. The proposed algorithm (ALR^n) is presented in Section 5. Work related to the our proposed algorithm is discussed in Section 6. We empirically evaluate the proposed algorithm in Section 7. We conclude in Section 8 with some pointers to future work.

2 Notation

We seek to assign a value $y \in \Omega_Y = \{y_1, \dots y_C\}$ of the class variable Y, to a given example $\mathbf{x} = (x_1, \dots, x_a)$, where the x_i are value assignments for the a features $\mathcal{A} = \{X_1, \dots, X_a\}$. We define $\binom{\mathcal{A}}{n}$ as the set of all subsets of $\mathcal{A}$ of size n, where each subset in the set is denoted as α: $\binom{\mathcal{A}}{n} = \{\alpha \subseteq \mathcal{A} : |\alpha| = n\}$. We use x_α to denote the set of values taken by features in the subset α for any data object $\mathbf{x}$.

LR for categorical data learns a weight for every feature value per class. Therefore, for LR, we denote, β_y to be the weight associated with class y, and β_{y,i,x_i} to be the weight associated with feature i taking value x_i with class label y. For LR^n, $\beta_{y,\alpha,x_\alpha}$ specifies the weight associated with class y and feature subset α taking value x_α. The equivalent weights for ALR^n are denoted by w_y, w_{y,i,x_i} and w_{y,α,x_α}. The probability of feature i taking value x_i given class y is denoted by $\text{P}(x_i \,|\, y)$. Similarly, probability of feature subset α, taking value x_α is denoted by $\text{P}(x_\alpha \,|\, y)$. Note, all probabilities are estimated probabilities. For clarity, we will not use $\hat{\text{P}}(.)$ notation which is typically used for estimated probabilities.

3 Higher-order Logistic Regression

LR is a linear classifier. For categorical features LR can be expressed as:

$$\text{P}_{LR}(y|\mathbf{x}) = \exp\Big(\beta_y + \sum_{i=1}^{a} \beta_{y,i,x_i} - \log \sum_{c \in \Omega_Y} \exp\Big(\beta_c + \sum_{j=1}^{a} \beta_{c,j,x_j}\Big)\Big). \quad (1)$$

Note, that LR for categorical data is often expressed as:

$$\mathrm{P}(y|\mathbf{x}) = \exp\Big(\beta_y + \sum_{i=1}^{a} \beta_{y,i,x_i} \mathbf{1}(X_i = x_i, Y = y) - \log \sum_{c \in \Omega_Y} \exp\Big(\beta_c + \sum_{j=1}^{a} \beta_{c,j,x_j} \mathbf{1}(X_j = x_j, Y = c)\Big)\Big), \quad (2)$$

where $\mathbf{1}(\cdot)$ is an indicator function that is 1 if it satisfies the input condition and zero otherwise. (2) reformulates (1) to sum only over the values that the indicator function will not cancel out.

Because its model is linear, LR is very restricted in the posterior distributions that it can precisely model. For example, it cannot model exclusive-or (XOR) between two features.

Adding *higher-order* features to LR increases the range of distributions that it can precisely model. Here, we define *higher-order* categorical features as features that are the Cartesian product of the primitive features, where the order n is the number of primitive features in the Cartesian product.

As mentioned in Section 1, it has been shown that Bayesian Network Classifiers that explicitly represent higher-order features tend to have lower bias than those that do not, and that the bias decreases as the order of the features increases [21,37]. Therefore, it seems likely that LR applied to higher-order features will likewise tend to have lower bias, with bias decreasing as the order increases. This is very significant, as LR is a powerful learning system and there is good reason to believe that the lower the bias of a learning system the lower its error will tend to be on very large datasets [3].

We define LR^n as:

$$\mathrm{P}_{LR^n}(y|\mathbf{x}) = \frac{\exp\big(\beta_y + \sum_{\alpha \in \binom{A}{n}} \beta_{y,\alpha,x_\alpha}\big)}{\sum_{c \in \Omega_Y} \exp\big(\beta_c + \sum_{\alpha^* \in \binom{A}{n}} \beta_{c,\alpha^*,x_{\alpha^*}}\big)}. \quad (3)$$

Again, we are expressing the definition in a non-standard form for the sake of clarity. The conventional definition is:

$$\mathrm{P}_{LR^n}(y|\mathbf{x}) = \frac{\exp\big(\beta_y + \sum_{\alpha \in \binom{A}{n}} \beta_{y,\alpha,x_\alpha} \mathbf{1}(X_\alpha = x_\alpha, Y = y)\big)}{\sum_{c \in \Omega_Y} \exp\big(\beta_c + \sum_{\alpha^* \in \binom{A}{n}} \beta_{c,\alpha^*,x_{\alpha^*}} \mathbf{1}(X_{\alpha^*} = x_{\alpha^*}, Y = c)\big)}.$$

Note, that, in this work, we do not include lower-order terms. For example, if $n = 2$ we do not include terms for β_{y,i,x_i}, because doing so does not increase the space of distinct distributions that can be modeled but does increase the number of parameters that must be optimized. However, it should be noted that including lower-level terms with regularization will affect the learning process and hence the model learned. A further advantage of including lower-order terms is that it provides an elegant backtracking procedure. If an higher-order term is not present at the training time but only appears at classification time, one can use the lower-order weights instead. In the current formulation,

there is no such backtracking mechanism. As will be discussed in Section 8, this hierarchical parameterization of ALR^n has been left as a promising direction for future work. We also note that, LR^n can be viewed as a Generalized Linear Model with a logistic *link* function and a fractional factorial design [14].

3.1 Kernel LR and LR^n

One way to deal with non-linearities in the data when applying plain Logistic Regression (LR) is by using kernels. Popularized with the advent of Support Vector Machines (SVM) using the kernel trick [1], one can project the data into a higher dimensional space without explicitly making the transformation. We can always write LR in the following form: $\mathrm{P}(y|\mathbf{x}) = \frac{1}{1+\exp(\beta^T\phi(\mathbf{x}))}$,where $\phi(\mathbf{x})$ is some function. By virtue of the representer theorem, we can write the β vector as: $\beta = \sum_i \alpha_i \phi(\mathbf{x})$, which leads to LR of the form:

$$\begin{aligned}\mathrm{P}(y|\mathbf{x}) &= \frac{1}{1+\exp(\sum_i \alpha_i \phi(\mathbf{x}_i)\phi(\mathbf{x}))},\\ &= \frac{1}{1+\exp(\sum_i \alpha_i k(\mathbf{x}_j,\mathbf{x}))}.\end{aligned} \tag{4}$$

Equation 4 represents a form of higher-order LR with kernels. Several kernels can be used such as linear, Gaussian and sigmoid. Of particular relevance is a polynomial kernel of degree d that takes the form: $k(u,v) = (u.v)^d$ that includes d-degree terms in LR. $d = 1$ leads to LR^1, $d = 2$ leads to LR^2 and so on. A form similar to this is used by SVM, but SVMs have an advantage that most of α_i are zeros. This is due to the loss function that SVM optimizes – Hinge Loss. Those α_i that are not zero are known as the support vectors. However, LR's log loss function does not lead to such sparsity. The non-sparse nature of KLR is one of its biggest drawback. Several methods such as Import Vector Machines (IVM) have been proposed to address this drawback and make KLR scalable to larger quantities of data [42,18,29,38]. The computational complexity of KLR is $\mathcal{O}(N^3)$ and is not suitable for big data classification since it will not be any faster than the k-nearest neighbour classifier. Even sparse models such as SVM are not suitable for big data and truly large scale applications as they can also suffer from the *curse of supporting vectors*, i.e., most of α_i are non-zero [30]. Kernel machines (e.g., IVM, SVM, etc.) are not substantially faster than k-nearest neighbour classifiers, as the number of support vectors are linear in the training set size [31].

3.2 Experimental Evaluation of LR^n

While LR^n is part of established data analytics practice, we are not aware of any research into its bias/variance profile or its performance relative to standard LR with respect to varying quantities of data. We here investigate those issues. Though we provide a detailed empirical analysis in Section 7,

here we present some results to illustrate the power of modeling higher-order interactions.

Figure 2 shows learning curves for LR^n with $n = 1, 2$ and 3. We generated these curves using a prequential testing paradigm on the `Localization` dataset. For each run, we first randomized the dataset. Then the ordered dataset was processed sequentially. Each example was first classified and the probabilistic loss: $\frac{1}{C}\sum_c^C(\delta_{y=c} - \mathrm{P}(c|\mathbf{x}))^2$, where $\delta_{y=c}$ is an indicator function that is 1 if the actual class label y is the same as c and zero otherwise, is calculated. Then the example is used to update the model. This process was repeated five times with different randomization of the dataset. For each run this process generated N loss values, where $N = 164860$, the size of the `Localization` dataset. To generate learning curves, for each point i on the X-axis, we plot: $\frac{1}{T}\sum_{k=\max(i-T,1)}^{i}\mathrm{loss}(x_k)$, where T is set to 1000. For $T \leq 1000$, we plot $\frac{1}{i}\sum_{k=1}^{i}\mathrm{loss}(x_k)$. It can be seen that for very small data quantity the lower variance LR^2 results in lower error than LR^3, but as data quantity increases the lower bias of LR^3 results in lowest error. It can be seen that LR obtains better performance than LR^2 and LR^3 when learned from very small quantities of data (the learning curves are zoomed in between 0 and 1000 instances in Figure 2 to illustrate this point). The obvious reason for the poor performance of LR^2 and LR^3 (models that including higher-order interactions) on smaller training sets is due to over-fitting. The powerful models can fit chance regularities in the data. Hence for smaller quantities of data, some sort of regularization that pulls the weights for many higher-order interactions back towards zero would lead to much better performance.

We note when interpreting results presented on insufficient data (as is the case for the bottom plot in Figure 2) it is easy for a data analyst to be misled into believing that the curves are diverging and that the higher-order classifier (LR^3) will asymptote to poorer performance than the lower-order classifier (LR^2) on large data – a misunderstanding that is due to the faster learning rate that is achieved initially by the lower-order classifier.

Figure 3, shows a comparative scatter of *Bias* of LR^n as n increases (we compare LR^1 with LR^2 and LR^2 with LR^3, where LR^1 is the standard LR). It can be seen that on the majority of 75 datasets from UCI repository (Table 1), the higher the value of n the lower the bias of LR^n. The results are based on two rounds of two-fold cross-validation. In line with our expectation that low inductive bias will often lead to low statistical bias which will in turn translate to lower error on big datasets, it can be seen that in Figure 4, higher-order LR results in much lower 0-1 Loss than standard LR and that this benefit tends to continue as n increases. Note that for one dataset, `poker-hand` [2], LR^2 achieves much lower error than LR^3 – we conjecture, that this is because of strong two-level correlations that exists in the data. On this synthetic (and deterministic)

[2] Dataset is about classifying poker hands (each hand constitutes five cards) into 10 different classes, i.e., one pair, two pair, three of a kind, straight, flush, full house, four of a kind, straight flush, royal flush and nothing in hand. Each card is represented by two attributes that is card suite and card number. Therefore, there are total of 10 attributes describing a hand.

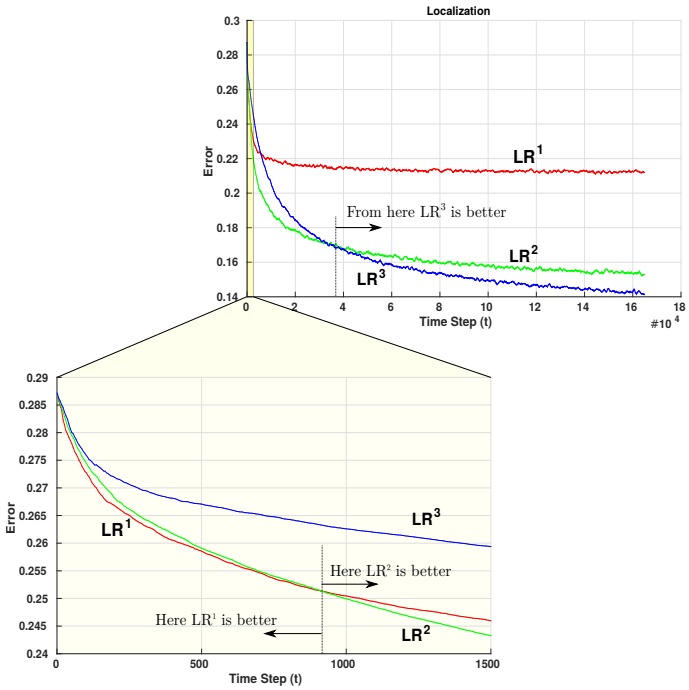

Fig. 2: Comparison of the performance (RMSE) of LR^1, LR^2 and LR^3 with varying quantities of data. For this demonstration, we used Stochastic gradient descent (SGD) for training the parameters of each model.

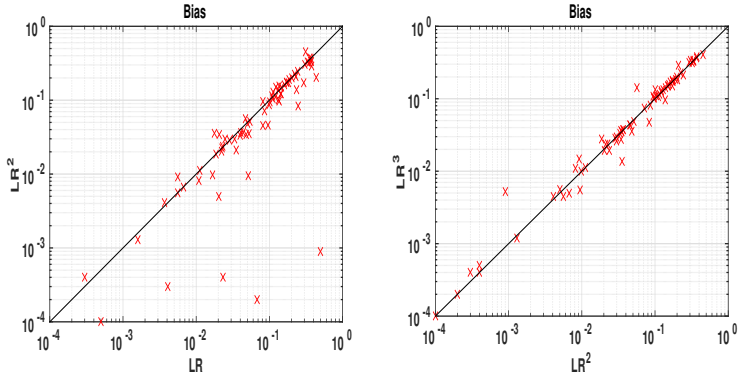

Fig. 3: Comparison of the scatter of the bias performance on 75 different datasets of LR vs. LR^2 (Left) and LR^2 vs. LR^3 (Right).

dataset, LR^3 will need much more data to estimate its parameters effectively. The current results only utilize half of the training data. It can be seen that for LR^2 this much data is more than enough but not for LR^3 and hence, LR^3 results in poor performance than LR^2.

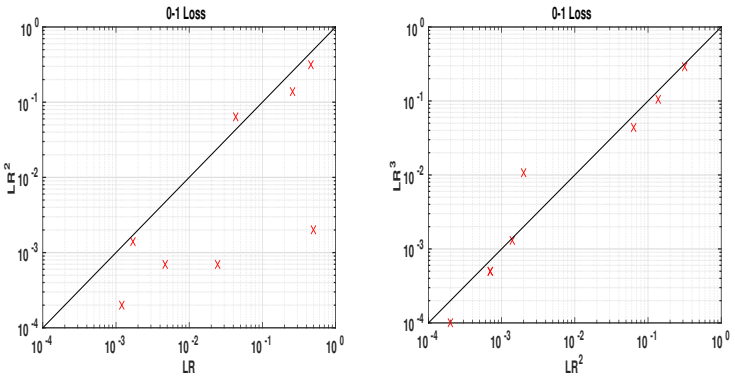

Fig. 4: Comparison of the scatter of 0-1 Loss on 9 *Big* (> 100000 instances) datasets of LR vs. LR^2 (Left) and LR^2 vs. LR^3 (Right).

4 Using Generative Models to Precondition Discriminative Learning

It has been shown that a direct equivalence between a weighted NB and LR can be exploited to greatly speed up LR's learning rate [39,40].

NB can be expressed as:

$$\mathrm{P}_{NB}(y|\mathbf{x}) = \frac{\mathrm{P}(y)\prod_{i=1}^{a}\mathrm{P}(x_i|y)}{\sum_{c\in\Omega_Y}\mathrm{P}(c)\prod_{i=1}^{a}\mathrm{P}(x_i|c)}. \tag{5}$$

One can add weights to NB to alleviate the feature independence assumption, resulting in the WANBIA-C formulation [39], that can be written as:

$$\begin{aligned}\mathrm{P}_{W}(y|\mathbf{x}) &= \frac{\mathrm{P}(y)^{w_y}\prod_{i=1}^{a}\mathrm{P}(x_i|y)^{w_{y,i,x_i}}}{\sum_{c\in\Omega_Y}\mathrm{P}(c)^{w_c}\prod_{j=1}^{a}\mathrm{P}(x_i|c)^{w_{c,j,x_j}}} \\ &= \exp\Big(w_y\log\mathrm{P}(y)+\sum_{i=1}^{a}w_{y,i,x_i}\log\mathrm{P}(x_i|y)- \\ &\qquad \log\sum_{c\in\Omega_Y}\exp\Big(w_c\log\mathrm{P}(c)+\sum_{j=1}^{a}w_{c,j,x_j}\log\mathrm{P}(x_j|c)\Big)\Big). \end{aligned}\tag{6}$$

When conditional log-likelihood (CLL) is maximized for LR and weighted NB using Equation 1 and 6 respectively, we get an equivalence such that $\beta_c \propto w_c \log \mathrm{P}(c)$ and $\beta_{c,i,x_i} \propto w_{c,i,x_i}\log\mathrm{P}(x_i|c)$. Thus, WANBIA-C and LR generate equivalent models.

While it might seem less efficient to use WANBIA-C, which has twice the number of parameters of LR, the probability estimates are learned very efficiently using maximum likelihood estimation, and provide useful information about the classification task that in practice serve to effectively precondition the search for the parameterization of weights to maximize conditional log-likelihood [39].

5 Accelerated Logistic Regression (ALR)

In order to create an efficient and effective low-bias learner, we want to perform the same strategy that is used by WANBIA-C for LR with higher-order categorical features. To precondition such a model using generative learning, we would like to build a model of the form:

$$\mathrm{P}(y\,|\,\mathbf{x}) = \frac{\mathrm{P}(y)\prod_{\alpha\in\binom{\mathcal{A}}{n}}\mathrm{P}(x_\alpha\,|\,y)}{\sum_{c\in\Omega_Y}\Big(\mathrm{P}(c)\prod_{\alpha^*\in\binom{\mathcal{A}}{n}}\mathrm{P}(x_{\alpha^*}\,|\,c)\Big)} \tag{7}$$

$$= \exp\Big(\log\mathrm{P}(y) + \sum_{\alpha\in\binom{\mathcal{A}}{n}}\log\mathrm{P}(x_\alpha\,|\,y) - \log\sum_{c\in\Omega_Y}\exp\Big(\log\mathrm{P}(c) + \sum_{\alpha^*\in\binom{\mathcal{A}}{n}}\log\mathrm{P}(x_{\alpha^*}\,|\,c)\Big)\Big). \tag{8}$$

The only existing generative model of this form is a log-linear model, which requires computationally expensive conditional log-likelihood optimization and consequently would not be efficient to employ. It is not possible to create a Bayesian network of this form as it would require that $\mathrm{P}(x_i, x_j)$ be independent of $\mathrm{P}(x_i, x_k)$, which is impossible because they share the common feature x_i. However, we can use a variant of the AnDE [37,36] approach of averaging many Bayesian networks. Unlike AnDE, we cannot use the arithmetic mean of the probability estimates from the constituent models, as we require a product of terms in numerator of Equation 7 rather than a sum, so we must instead use a geometric mean.

5.1 Averaged n-Join Estimators (AnJE)

Let $\mathcal{P}$ be a partition of the features $\mathcal{A}$. By assuming independence only between the sets of features $A \in \mathcal{P}$ one obtains an n-join estimator:

$$\mathrm{P}_{\mathrm{AnJE}}(\mathbf{x}\,|\,y) = \prod_{\alpha\in\mathcal{P}}\mathrm{P}(x_\alpha\,|\,y).$$

For example, if there are four features X_1, X_2 , X_3 and X_4 that are partitioned into the sets $\{X_1, X_2\}$ and $\{X_3, X_4\}$ then by assuming conditional independence between the sets we obtain

$$\mathrm{P}_{\mathrm{AnJE}}(x_1, x_2, x_3, x_4\,|\,y) = \mathrm{P}(x_1, x_2\,|\,y)\mathrm{P}(x_3, x_4\,|\,y).$$

Let $\Psi_n^{\mathcal{A}}$ be the set of all partitions of $\mathcal{A}$ such that $\forall_{\mathcal{P}\in\Psi_n^{\mathcal{A}}}\forall_{\alpha\in\mathcal{P}}|\alpha| = n$. For convenience we assume that $|\mathcal{A}|$ is a multiple of n. Let $\Upsilon_N^{\mathcal{A}}$ be a subset of $\Psi_n^{\mathcal{A}}$ that includes each set of n features once,

$$\Upsilon_N^{\mathcal{A}} \subseteq \Psi_n^{\mathcal{A}} : \forall_{\alpha\in\binom{\mathcal{A}}{n}}\big|\{\mathcal{P}\in\Upsilon_N^{\mathcal{A}} : \alpha\in\mathcal{P}\}\big| = 1.$$

The AnJE model is the geometric mean of the set of n-join estimators for the partitions $Q \in \Upsilon_N^{\mathcal{A}}$. Note, that this partitioning of features can be viewed from combinatorial design theory's perspective, where the idea of partitioning the space is commonly used to reduce the overall complexity of the problem [32, 19].

The AnJE estimate of conditional likelihood on a per-datum-basis can be written as:

$$\mathrm{P}_{\mathrm{AnJE}}(y\,|\,\mathbf{x}) \propto \mathrm{P}(y)\mathrm{P}_{\mathrm{AnJE}}(\mathbf{x}|y) = \mathrm{P}(y) \prod_{\alpha \in \binom{\mathcal{A}}{n}} \mathrm{P}(x_\alpha\,|\,y)^{\frac{(n-1)!(a-n)!}{(a-1)!}}. \tag{9}$$

This is derived as follows. Each $\mathcal{P}$ is of size $s = a/n$. There are $\binom{a}{n}$ feature-value n-tuples. Each must occur in exactly one partition, so the number of partitions must be

$$p = \binom{a}{n}/s = \frac{(a-1)!}{(n-1)!(a-n)!}. \tag{10}$$

The geometric mean of all the AnJE models is thus

$$\mathrm{P}_{\mathrm{AnJE}}(\mathbf{x}\,|\,y) = \sqrt[p]{\prod_{\alpha \in \binom{\mathbf{x}}{a}} \mathrm{P}(x_\alpha\,|\,y)} = \prod_{\alpha \in \binom{\mathcal{A}}{a}} \mathrm{P}(x_\alpha\,|\,y)^{(n-1)!(a-n)!/(a-1)!}. \tag{11}$$

Using Equation 9, we can write the log of $\mathrm{P}(y|\mathbf{x})$ as:

$$\log \mathrm{P}_{\mathrm{AnJE}}(y\,|\,\mathbf{x}) \propto \log \mathrm{P}(y) + \frac{(n-1)!(a-n)!}{(a-1)!} \sum_{\alpha \in \binom{\mathcal{A}}{a}} \log \mathrm{P}(x_\alpha\,|\,y). \tag{12}$$

5.2 ALR^n

It can be seen that AnJE is a simple model that places the weight defined in Equation 10 on all feature subsets in the ensemble. The main advantage of this weighting scheme is that it requires no optimization, making AnJE learning extremely efficient. All that is required for training is to calculate the counts from the data. However, the disadvantage of AnJE is its inability to perform any form of discriminative learning. Our proposed algorithm, ALR^n uses AnJE to precondition LR^n by placing weights on all probabilities in Equation 7 and learning these weights by optimizing the conditional-likelihood. One can, however, initialize these weights with weights in Equation 10 for faster convergence. We will discuss this in Appendix B. One can re-write AnJE models with this parameterization as:

$$\begin{aligned}\mathrm{P}_{\mathrm{ALR}}(y\,|\,\mathbf{x}) = \exp\Big(w_y \log \mathrm{P}(y) + \sum_{\alpha \in \binom{\mathcal{A}}{n}} w_{y,\alpha,x_\alpha} \log \mathrm{P}(x_\alpha\,|\,y) - \\ \log \sum_{c \in \Omega_Y} \exp\Big(w_c \log \mathrm{P}(c) + \sum_{\alpha^* \in \binom{\mathcal{A}}{n}} w_{c,\alpha^*,x_{\alpha^*}} \log \mathrm{P}(x_{\alpha^*}\,|\,c)\Big)\Big).\end{aligned} \tag{13}$$

Note that we can compute the likelihood and class-prior probabilities using either MLE or MAP. Therefore, we can write Equation 22 as:

$$\log \mathrm{P}_{\mathrm{ALR}}(y\,|\,\mathbf{x}) = w_y \log \pi_y + \sum_{\alpha \in \binom{\mathcal{A}}{n}} w_{y,\alpha,x_\alpha} \log \theta_{x_\alpha\,|\,y} - \log \sum_{c \in \Omega_Y} \exp\Big(w_c \log \pi_c + \sum_{\alpha^* \in \binom{\mathcal{A}}{n}} w_{c,\alpha^*,x_{\alpha^*}} \log \theta_{x_{\alpha^*}\,|\,c} \Big). \quad (14)$$

Assuming a Dirichlet prior, a MAP estimate of $\mathrm{P}(y)$ is π_y which equals: $\frac{\#_y + m/|\mathcal{Y}|}{t+m}$, where $\#_y$ is the number of instances in the dataset with class y and t is the total number of instances, and m is the smoothing parameter. We will set $m = 1$ in this work. Similarly, a MAP estimate of $\mathrm{P}(x_\alpha\,|\,y)$ is $\theta_{x_\alpha|y}$ which equals: $\frac{\#_{x_\alpha,y} + m/|x_\alpha|}{\#_y + m}$, where $\#_{x_\alpha,y}$ is the number of instances in the dataset with class y and feature values x_α.

ALR^n computes weights by optimizing CLL. Therefore, one can compute the gradient of Equation 14 with-respect-to weights and rely on gradient descent based methods to find the optimal value of these weights. Since we do not want to be stuck in local minimums, a natural question to ask is whether the resulting objective function is convex [2]. It turns out that the objective function of ALR^n is indeed convex. [27] proved that an objective function of the form $\sum_{\mathbf{x} \in \mathcal{D}} \log \mathrm{P}_{\mathcal{B}}(y|\mathbf{x})$, optimized by any conditional Bayesian network model is convex if and only if the structure $\mathcal{G}$ of the Bayesian network $\mathcal{B}$ is perfect. A directed graph in which all nodes having a common child are connected is called perfect [17]. ALR^n is a geometric mean of several sub-models where each sub-model models $\lfloor \frac{a}{n} \rfloor$ interactions each conditioned on the class feature. Each sub-model has a structure that is perfect. Since, the product of two convex objective functions leads to a convex function, one can see that ALR^n's optimization function will also lead to a convex objective function.

Let us first calculate the gradient of Equation 14 with-respect-to weights associated with π_y. We can write:

$$\frac{\partial \log \mathrm{P}(y\,|\,\mathbf{x})}{\partial w_y} = \mathbf{1}_y \log \pi_y - \frac{\pi_y^{w_y} \log \pi_y \prod_{\alpha \in \binom{\mathcal{A}}{n}} \theta_{x_\alpha\,|\,y}^{w_{y,\alpha,x_\alpha}}}{\sum_{c \in \Omega_Y} \pi_c^{w_c} \prod_{\alpha^* \in \binom{\mathcal{A}}{n}} \theta_{x_\alpha^*\,|\,c}^{w_{c,\alpha^*,x_{\alpha^*}}}} = (\mathbf{1}_y - \mathrm{P}(y|\mathbf{x})) \log \pi_y, \quad (15)$$

where $\mathbf{1}_y$ denotes an indicator function that is 1 if derivative is taken with-respect-to class y and 0 otherwise. Computing the gradient with-respect-to weights associated with $\theta_{x_\alpha|y}$ gives:

$$\frac{\partial \log \mathrm{P}(y\,|\,\mathbf{x})}{\partial w_{y,\alpha,x_\alpha}} = \mathbf{1}_y \mathbf{1}_\alpha \log \theta_{x_\alpha|y} - \frac{\pi_y^{w_y} \prod_{\alpha \in \binom{\mathcal{A}}{n}} \theta_{x_\alpha\,|\,y}^{w_{y,\alpha,x_\alpha}} \mathbf{1}_\alpha \log \theta_{x_\alpha|y}}{\sum_{c \in \Omega_Y} \pi_c^{w_c} \prod_{\alpha^* \in \binom{\mathcal{A}}{n}} \theta_{x_{\alpha^*}\,|\,c}^{w_{c,\alpha^*,x_{\alpha^*}}}} = (\mathbf{1}_y - \mathrm{P}(y|\mathbf{x})) \mathbf{1}_\alpha \log \theta_{x_\alpha|y}, \quad (16)$$

where $\mathbf{1}_\alpha$ and $\mathbf{1}_y$ denotes an indicator function that is 1 if the derivative is taken with-respect-to feature set α (respectively, class y) and 0 otherwise.

5.3 Alternative Parameterization

Let us reparameterize ALR^n such that:

$$\beta_y = w_y \log \pi_y, \quad \text{and} \quad \beta_{y,\alpha,x_\alpha} = w_{y,\alpha,x_\alpha} \log \theta_{x_\alpha|y}. \tag{17}$$

Now, we can re-write Equation 14 as:

$$\log \mathrm{P}_{LR}(y|\mathbf{x}) = \beta_y + \sum_{\alpha \in \binom{\mathcal{A}}{n}} \beta_{y,\alpha,x_\alpha} - \log \sum_{c \in \Omega_Y} \exp\Big(\beta_c + \sum_{\alpha^* \in \binom{\mathcal{A}}{n}} \beta_{c,\alpha^*,x_{\alpha^*}}\Big). \tag{18}$$

It can be seen that this leads to Equation 3. We call this parameterization LR^n.

Like ALR^n, LR^n also leads to a convex optimization problem, and, therefore, its weights can also be optimized by simple gradient decent based algorithms. Let us compute the gradient of objective function in Equation 18 with-respect-to β_y. In this case, we can write:

$$\frac{\partial \log \mathrm{P}(y \,|\, \mathbf{x})}{\partial \beta_y} = (\mathbf{1}_y - \mathrm{P}(y|\mathbf{x})). \tag{19}$$

Similarly, computing gradient with-respect-to $\beta_{y,\alpha,x_\alpha}$, we can write:

$$\frac{\partial \log \mathrm{P}(y \,|\, \mathbf{x})}{\partial \beta_{y,\alpha,x_\alpha}} = (\mathbf{1}_y - \mathrm{P}(y|\mathbf{x}))\mathbf{1}_\alpha. \tag{20}$$

5.4 Comparative analysis of ALR^n and LR^n

It can be seen that the two models are actually equivalent and each is a reparameterization of the other. However, there are subtle distinctions between the two. The most important distinction is the utilization of MAP or MLE probabilities in ALR^n. Therefore, ALR^n is a two step learning algorithm:

- Step 1 is the optimization of the log-likelihood of the data ($\log \mathrm{P}(y, \mathbf{x})$) to obtain the estimates of the prior and likelihood probabilities. One can view this step as of *generative learning*.
- Step 2 is the introduction of weights on these probabilities and learning of these weights by maximizing CLL ($\mathrm{P}(y \,|\, \mathbf{x})$) objective function. This step can be interpreted as *discriminative learning*.

ALR^n employs generative-discriminative learning as opposed to only discriminative learning by LR^n.

One can expect a similar bias-variance profile and a very similar classification performance as both models will converge to a similar point in the optimization space, the only difference in the final parameterization being due to recursive descent being terminated before absolute convergence. However, the rate of convergence of the two models can be very different. [39] shows that for

NB, such ALR^n style parameterization with generative-discriminative learning can greatly speed-up convergence relative to only discriminative training. Note, discriminative training with NB as the graphical model is vanilla LR. We expect to see the same trend in the convergence performance of ALR^n and LR^n.

Another distinction between the two models becomes explicit if a regularization penalty is added to the objective function. One can see that in case of ALR^n, regularizing weights towards 1 will effectively pull parameters back towards the generative training estimates. For smaller datasets, one can expect to obtain better performance by using a large regularization parameter and pulling estimates back towards 1. However, one cannot do this for LR^n. Therefore, ALR^n models can very elegantly combine generative and discriminative parameters.

An analysis of the gradient of ALR^n in Equation 15 and 16 and that of LR^n in Equation 19 and 20 also reveals an interesting comparison. We can write ALR^n's gradients in terms of LR^n's gradient as follows:

$$\begin{aligned}\frac{\partial \log \mathrm{P}(y \mid \mathbf{x})}{\partial w_y} &= \frac{\partial \log \mathrm{P}(y \mid \mathbf{x})}{\partial \beta_y} \log \pi_y, \\ \frac{\partial \log \mathrm{P}(y \mid \mathbf{x})}{\partial w_{y,\alpha,x_\alpha}} &= \frac{\partial \log \mathrm{P}(y \mid \mathbf{x})}{\partial \beta_{y,\alpha,x_\alpha}} \log \theta_{x_\alpha|y}. \end{aligned} \tag{21}$$

It can be seen that ALR^n has the effect of re-scaling LR^n's gradient by the log of the conditional probabilities. We conjecture that such re-scaling has the effect of pre-conditioning the parameter space and, therefore, will lead to faster convergence.

6 Related Work

Averaged n-Dependent Estimators (AnDE) is the inspiration for AnJE. An AnDE model is the arithmetic mean of all Bayesian Network Classifiers in each of which all features depend on the class and the some n features. A simple depiction of A1DE in graphical form in shown in Figure 5. There are $\binom{a}{n}$ possible combination of features that can be used as parents, producing $\binom{a}{n}$ sub-models which are combined by averaging.

AnDE and AnJE both use simple generative learning, merely the counting the relevant *sufficient statistics* from the data. Both have only one tweaking parameter: n – that controls the bias-variance trade-off. Higher values of n leads to low bias and high variance and vice-versa.

It is important not to confuse the equivalence (in terms of the level of interactions they model) of AnJE and AnDE models. That is, the following

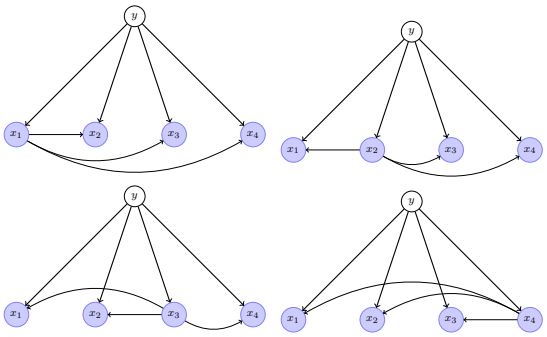

Fig. 5: Sub-models in an AnDE model with $n = 2$ and with four features.

holds:

$$\begin{aligned} f(\text{A2JE}) &= f(\text{A1DE}), \\ f(\text{A3JE}) &= f(\text{A2DE}), \\ \vdots \quad &= \quad \vdots \\ f(\text{AnJE}) &= f(\text{A(n-1)DE}), \end{aligned}$$

where $f(.)$ is a function that returns the number of interactions that the algorithm models. Also, A1JE = A0DE = naive Bayes. Thus, an AnJE model uses the same core statistics as an A(n-1)DE model. At training time, AnJE and A(n-1)DE must learn the same information from the data. However, at classification time, each of these statistics is accessed once by AnJE and n times by A(n-1)DE, making AnJE more efficient. However, as we will show, it turns out that AnJE's use of the geometric mean results in a more biased estimator than the arithmetic mean used by AnDE. As a result, in practice, an AnJE model is less accurate than the equivalent AnDE model. However, due to the use of the arithmetic mean by AnDE, its weighted version would be much more difficult to optimize than AnJE, as when transformed to log space it does not admit to a simple linear model.

ALR^n has a number of similarities with *ELR* [12,11] for which the parameters associated with a Bayesian network classifier (naive Bayes or TAN) are learned by optimizing the CLL. ELR performs discriminative learning of the weights for a model with a Bayesian network structure. As explained in Section 5, it is not possible to create a single Bayesian network with the structure of the ALR model. Further, ELR does not utilize the generative parameters to precondition the search for discriminative parameters as does ALR. Some related ideas to *ELR* are also explored in [25,26,33].

Feature construction has been studied extensively [20]. The goal is to improve the classifier's accuracy by creating new attributes from existing attributes. The new attributes can be either binary or arithmetic or other combinations of existing attributes. One approach that is closely related to the current work is the formation of Cartesian products of categorical features

Domain	Case	Att	Class	Domain	Case	Att	Class
Kddcup	5209000	41	40	Vowel	990	14	11
Poker-hand	1175067	10	10	Tic-Tac-ToeEndgame	958	10	2
MITFaceSetC	839000	361	2	Annealing	898	39	6
Covertype	581012	55	7	Vehicle	846	19	4
MITFaceSetB	489400	361	2	PimaIndiansDiabetes	768	9	2
MITFaceSetA	474000	361	2	BreastCancer(Wisconsin)	699	10	2
Census-Income(KDD)	299285	40	2	CreditScreening	690	16	2
Localization	164860	7	3	BalanceScale	625	5	3
Connect-4Opening	67557	43	3	Syncon	600	61	6
Statlog(Shuttle)	58000	10	7	Chess	551	40	2
Adult	48842	15	2	Cylinder	540	40	2
LetterRecognition	20000	17	26	Musk1	476	167	2
MAGICGammaTelescope	19020	11	2	HouseVotes84	435	17	2
Nursery	12960	9	5	HorseColic	368	22	2
Sign	12546	9	3	Dermatology	366	35	6
PenDigits	10992	17	10	Ionosphere	351	35	2
Thyroid	9169	30	20	LiverDisorders(Bupa)	345	7	2
Pioneer	9150	37	57	PrimaryTumor	339	18	22
Mushrooms	8124	23	2	Haberman'sSurvival	306	4	2
Musk2	6598	167	2	HeartDisease(Cleveland)	303	14	2
Satellite	6435	37	6	Hungarian	294	14	2
OpticalDigits	5620	49	10	Audiology	226	70	24
PageBlocksClassification	5473	11	5	New-Thyroid	215	6	3
Wall-following	5456	25	4	GlassIdentification	214	10	3
Nettalk(Phoneme)	5438	8	52	SonarClassification	208	61	2
Waveform-5000	5000	41	3	AutoImports	205	26	7
Spambase	4601	58	2	WineRecognition	178	14	3
Abalone	4177	9	3	Hepatitis	155	20	2
Hypothyroid(Garavan)	3772	30	4	TeachingAssistantEvaluation	151	6	3
Sick-euthyroid	3772	30	2	IrisClassification	150	5	3
King-rook-vs-king-pawn	3196	37	2	Lymphography	148	19	4
Splice-junctionGeneSequences	3190	62	3	Echocardiogram	131	7	2
Segment	2310	20	7	PromoterGeneSequences	106	58	2
CarEvaluation	1728	8	4	Zoo	101	17	7
Volcanoes	1520	4	4	PostoperativePatient	90	9	3
Yeast	1484	9	10	LaborNegotiations	57	17	2
ContraceptiveMethodChoice	1473	10	3	LungCancer	32	57	3
German	1000	21	2	Contact-lenses	24	5	3
LED	1000	8	10				

Table 1: Details of Datasets

through hill-climbing search [24]. Our work differs in using all Cartesian products of a given order and using discriminative learning of weights to determine each combinations relative (weighted) contribution to the model.

7 Experiments

In this section, we compare and analyze the performance of our proposed algorithms and related methods on 76 natural domains from the UCI repository of machine learning datasets [8]. The experiments are conducted on the datasets described in Table 1. 40 datasets have fewer than $1,000$ instances, 20 datasets have between $1,000$ and $10,000$ instances and 16 datasets have more than $10,000$ instances. There are 8 datasets with over $100,000$ instances. These datasets are shown in bold font in Table 1.

Each algorithm is tested on each dataset using 5 rounds of 2-fold cross validation[3].

We compare four different metrics, i.e., 0-1 Loss, RMSE, Bias and Variance[4]. There are a number of different bias-variance decomposition definitions. In this research, we use the bias and variance definitions of [15] together with the repeated cross-validation bias-variance estimation method proposed by [35]. Kohavi and Wolpert [15] define bias and variance as follows:

$$\text{bias}^2 = \frac{1}{2}\sum_{y\in\mathcal{Y}}\Big(\mathrm{P}(y|\mathbf{x}) - \hat{\mathrm{P}}(y\,|\,\mathbf{x})\Big)^2,$$

and

$$\text{variance} = \frac{1}{2}\left(1 - \sum_{y\in\mathcal{Y}}\hat{\mathrm{P}}(y\,|\,\mathbf{x})^2\right).$$

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.

The datasets in Table 1 are divided into two categories. We call the following datasets *Big* – `KDDCup, Poker-hand, USCensus1990, Covertype, MITFaceSetB, MITFaceSetA, Census-income, Localization`. All remaining datasets are denoted as *Little* in the results.

Due to their size, experiments for most of the *Big* datasets had to be performed in a heterogeneous environment (grid computing) for which CPU wall-clock times are not commensurable. In consequence, when comparing classification and training time, the following 12 datasets constitutes *Big* category – `Localization, Census-income, Poker-hand, Covtype, Connect-4, Shuttle, Adult, Letter-recog, Magic, Nursery, Sign, Pendigits`. When comparing average results across *Little* and *Big* datasets, we normalize the results with respect to ALR^2 and present a geometric mean.

Numeric features are discretized by using the Minimum Description Length (MDL) supervised discretization method [6]. Training data is discretized at training time. The cut-points learned during the discretization procedure are used to discretize the testing data. However, for `kddcup`, `MITFaceSetA`, `MITFaceSetB`, `MITFaceSetC`, for computational efficiency, the entire dataset is discretized before the training starts. That is the cut-points are learned over both training and test data. The bias introduced by including test data in

[3] Exception is `MITFaceSetA`, `MITFaceSetB`, `MITFaceSetA` and `Kddcup` where results are reported with 2 rounds of 2-fold cross validation because of the time-constraints on the grid-computers on which the results were computed

[4] As discussed in Section 1, the reason for performing bias/variance estimation is that it provides insights into how the learning algorithm will 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 [3].

the discretization process is not an issue here because it is uniform across all compared classifiers (i.e., AnDE and Random Forest).

A missing value is treated as a separate feature value and taken into account exactly like other values.

We employed the L-BFGS quasi-Newton method for solving the optimization[5]. Note, that we have used L-BFGS to demonstrate the efficacy of ALR^n, the results generalize well to other optimization routines including Gradient Descent, Conjugate Gradient and Stochastic Gradient Descent (SGD). In Appendix A, we also present results with Conjugate Gradient optimization

Random Forest (RF) [4] is considered to be a state of the art classification scheme. It consist of multiple decision trees, each tree is trained on data selected at random but with replacement from the original data (bagging). For example, if there are N data points, select N data points at random with replacement. If there are a attributes, a number m is specified, such that $m < a$. At each node of the decision tree, m attributes are randomly selected out of a and are evaluated, the best being used to split the node. Note, we used $m = \log_2(a) + 1$, where a is the total number of features. Each tree is grown to its largest possible size and no pruning is done. An instance is classified by passing it through each decision tree and selecting the mode of the output of the decision trees. We used 100 decision trees in this work.

The Internal discretization mechanism of Random Forest is used for all but the `kddcup, MITFaceSetA, MITFaceSetB, MITFaceSetC` datasets, where the entire data is first discretized, as described before.

7.1 ALR^n vs. AnJE

A W-D-L comparison of the 0-1 Loss, RMSE, bias and variance of ALR^n and AnJE on *Little* datasets is shown in Table 2. We compare ALR^2 with A2JE and ALR^3 with A3JE only. It can be seen that ALR^n has significantly lower bias but significantly higher variance. The 0-1 Loss and RMSE results are not in favour of any algorithm. However, on *Big* datasets, ALR^n wins on 7 out of 8 datasets in terms of both RMSE and 0-1 Loss. The results are not significant since the p value of 0.070 is greater than our set threshold of 0.05. However, the evidence is consistent with the proposition that ALR^n successfully reduces the bias of AnJE, at the expense of increasing its variance.

Normalized 0-1 Loss and RMSE results for both models are shown in Figure 6. It can be seen that ALR^n has a lower averaged 0-1 Loss and RMSE than AnJE. This difference is substantial when comparing on *Big* datasets. The training and classification time of AnJE is, however, substantially lower than ALR^n as can be seen from Figure 7. This is to be expected as ALR^n adds discriminative training to AnJE and uses twice the number of parameters at classification time.

[5] The original L-BFGS implementation of [5] from `http://users.eecs.northwestern.edu/~nocedal/lbfgsb.html` is used.

	ALR^2 vs. A2JE		**ALR^3 vs. A3JE**	
	W-D-L	p	W-D-L	p
		All Datasets		
Bias	62/3/11	**<0.001**	55/9/12	**<0.001**
Variance	21/4/51	**<0.001**	25/2/49	**0.007**
		Little Datasets		
0-1 Loss	47/4/25	0.012	39/2/35	0.727
RMSE	39/0/37	0.908	32/0/44	0.206
		Big Datasets		
0-1 Loss	8/0/0	0.007	7/0/1	0.070
RMSE	8/0/0	0.007	7/0/1	0.070

Table 2: Win-Draw-Loss: ALR^2 vs. A2JE and ALR^3 vs A3JE. p is two-tail binomial sign test. Results are significant if $p \leq 0.05$.

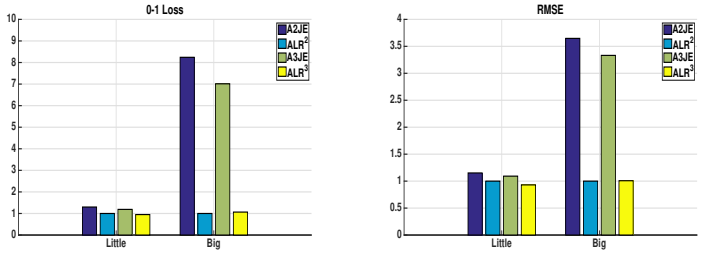

Fig. 6: Geometric mean of 0-1 Loss (Left), RMSE (Right) performance of ALR^2, A2JE, ALR^3 and A3JE for *Little* and *Big* datasets.

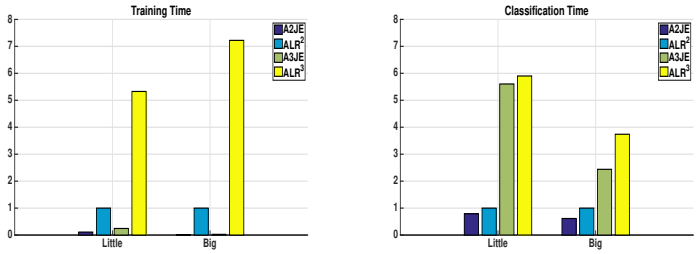

Fig. 7: Geometric mean of Training Time (Left), Classification Time (Right) of ALR^2, A2JE, ALR^3 and A3JE for *All* and *Big* datasets.

7.2 ALR^n vs. AnDE

A W-D-L comparison for 0-1 Loss, RMSE, bias and variance results of the two ALR^n models relative to the corresponding AnDE models are presented in Table 3. We compare ALR^2 with A1DE and ALR^3 with A2DE only. It can be seen that ALR^n has significantly lower bias and non-significantly higher variance than AnDE models. Recently, AnDE models have been proposed as a fast and effective Bayesian classifiers when learning from large quantities of data [41]. These bias-variance results make ALR^n a suitable alternative to

	ALR^2 vs. A1DE		**ALR^3 vs. A2DE**	
	W-D-L	p	W-D-L	p
		All Datasets		
Bias	60/5/11	**<0.001**	47/11/18	**<0.001**
Variance	22/9/45	0.006	26/4/46	0.024
		Little Datasets		
0-1 Loss	43/3/30	0.159	33/4/39	0.556
RMSE	30/0/46	0.084	24/0/52	0.035
		Big Datasets		
0-1 Loss	8/0/0	0.007	7/0/1	0.070
RMSE	8/0/0	0.073	7/0/1	0.070

Table 3: Win-Draw-Loss: ALR^2 vs. A1DE and ALR^3 vs A2DE. p is two-tail binomial sign test. Results are significant if $p \leq 0.05$.

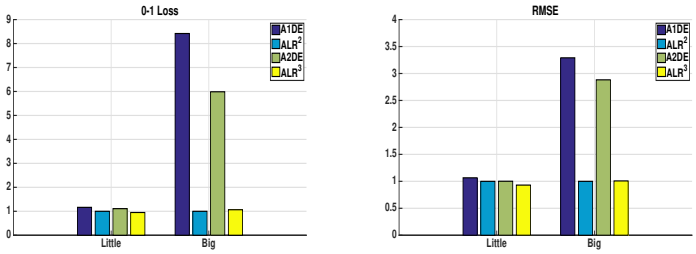

Fig. 8: Geometric mean of 0-1 Loss (Left) and RMSE (Right) performance of ALR^2, A1DE, ALR^3 and A2DE for *Little* and *Big* datasets.

AnDE when dealing with big data. The 0-1 Loss results are similar, but AnDE has better RMSE results than ALR^n on *Little* datasets. On *Big* datasets, it can be seen that ALR^n wins on majority of datasets.

Normalized 0-1 Loss and RMSE are shown in Figure 8. It can be seen that the ALR^n models have lower 0-1 Loss and RMSE than the corresponding AnDE models.

A comparison of the training time of ALR^n and AnDE is given in Figure 9. As expected, due to its additional discriminative learning, ALR^n requires substantially more training time than AnDE. However, AnDE does not share such a consistent advantage with respect to classification time, the relativities depending on the dimensionality of the data. For high-dimensional data the large number of permutations of features that AnDE must consider at classification time results in greater computation.

7.3 ALR^n vs. LR^n

In this section, we compare the two ALR^n models with their equivalent LR^n models. As discussed before, we expect to see similar bias-variance profiles and similar classification performance as the two models are re-parameterizations of each other.

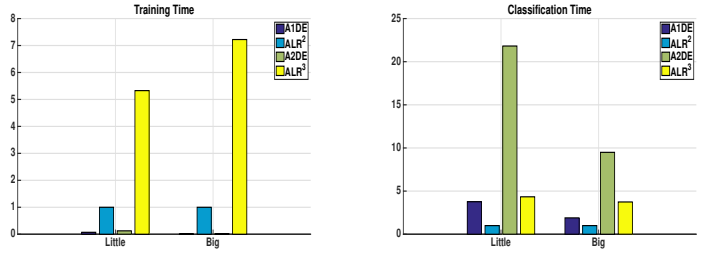

Fig. 9: Geometric mean of Training Time (Left), Classification Time (Right) of ALR^2, A1DE, ALR^3 and A2DE for *All* and *Big* datasets.

We compare the two parameterizations in terms of the scatter of their 0-1 Loss and RMSE values on *Little* datasets in Figure 10, 12 respectively, and on *Big* datasets in Figure 11, 13 respectively. It can be seen that the two parameterizations (with an exception of a few datasets[6]) have a similar spread of 0-1 Loss and RMSE values for both $n = 2$ and $n = 3$.

We attribute the difference in the performance of the two parameterizations in terms of 0-1 Loss due to the numerical instability of the solver. The L-BFGS library we are using is written in `java` that internally calls `C++` routines which eventually call a `fortran` library. There are some non-significant differences between LR^n and ALR^n only on the `Phoneme`, `Lung-cancer` and `Promoters` datasets. These models trained on these datasets are all extremely sparse. `Lung-cancer`, for example, has only 32 instances defined over 57 attributes and 3 classes. LR^2 and ALR^2 in this case optimize 75246 parameters and LR^3 and ALR^3 optimize 5465451 parameters. We conjecture that the difference in the performance (0-1 Loss) is due to over-flowing of the estimated parameters. It appears that on these datasets, data is linearly separable in spaces spanned by LR^2, ALR^2, LR^3 and ALR^3 – this leads to parameters becoming too large. For these datasets, ideally, one should regularize the two parameterizations differently (tuning λ on some validation set) to make sure that the parameter estimates do not get too low or too high.

The comparative scatter of the number of iterations each parameterization takes to converge is shown in Figure 14 and 15 for *Little* and *Big* datasets respectively. It can be seen that the number of iterations for ALR^n are far fewer than LR^n. It should be noted that the scatter plots are on the log-scale and the ratio such as: $\frac{10^{2.5}}{10^3}$ between ALR^n and LR^n results in three-times lesser iterations for ALR^n than LR^n.

The number of iterations to converge plays a major part in determining an algorithm's training time. The training time of the two parameterizations is shown in Figures 16 and 17 for *Little* and *Big* datasets, respectively. It can be seen that ALR^n models are much faster than the equivalent LR^n models.

[6] LR^2 vs. ALR^2: two datasets on which the 0-1 Loss of two parameterization is significantly different are: `Phoneme` (0.1935, 0.2814) and `Promoters` (0.1132, 0.1717). LR^3 vs. ALR^3: two datasets on which the 0-1 Loss of two parameterization is significantly different are: `Phoneme` (0.2347, 0.4743) and `Lung-Cancer` (0.6125, 0.5625).

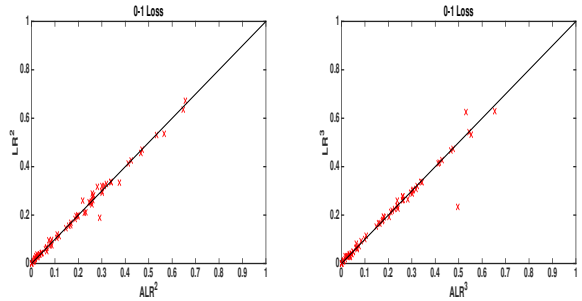

Fig. 10: Comparative scatter of 0-1 Loss of ALR^2 and LR^2 (Left) and ALR^3 and LR^3 (Right) for *Little* datasets.

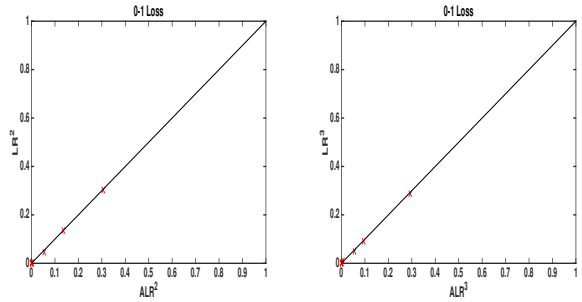

Fig. 11: Comparative scatter of 0-1 Loss of ALR^2 and LR^2 (Left) and ALR^3 and LR^3 (Right) for *Big* datasets.

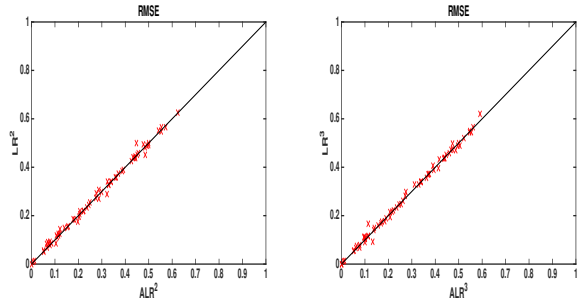

Fig. 12: Comparative scatter of RMSE of ALR^2 and LR^2 (Left) and ALR^3 and LR^3 (Right) for *Little* datasets.

Again, note that the scatter plots are on the log-scale. A simple ratio of $\frac{10^{5.6}}{10^{5.8}}$ between ALR^n and LR^n is difficult to distinguish as a point over the diagonal line in favour of ALR^n, but actually represents a speed-up of around 1.5 times.

A comparison of the rate of convergence of Negative-Log-Likelihood (NLL) of ALR^2 and LR^2 parameterizations on some sample datasets is shown in Figure 18. It can be seen that, ALR^2 has a steeper curve, asymptoting to its global minimum much quicker. For example, on almost all datasets, one can see that ALR^2 follows a steeper, hence more desirable, path toward convergence.

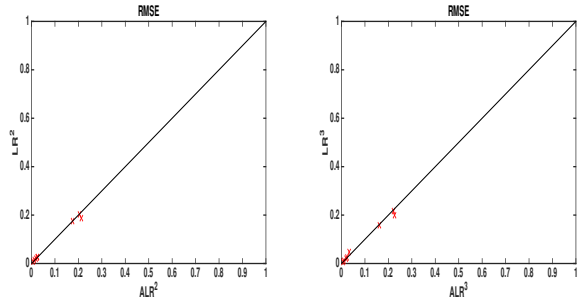

Fig. 13: Comparative scatter of RMSE of ALR^2 and LR^2 (Left) and ALR^3 and LR^3 (Right) for *Big* datasets.

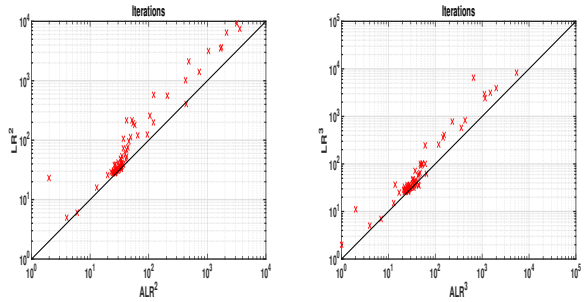

Fig. 14: Comparative scatter of number of iterations of ALR^2 and LR^2 (Left) and ALR^3 and LR^3 (Right) for *Little* datasets.

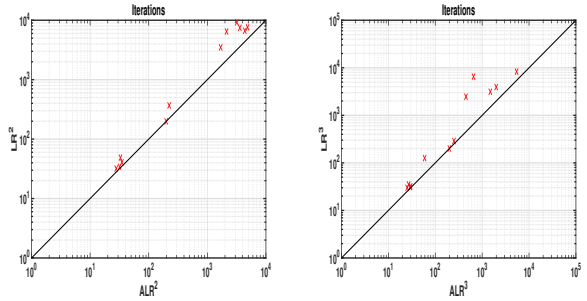

Fig. 15: Comparative scatter of iterations of ALR^2 and LR^2 (Left) and ALR^3 and LR^3 (Right) for *Big* datasets.

This is extremely advantageous when learning from very few iterations (for example, when learning using Stochastic Gradient Descent based optimization) and, therefore, is a desirable property for scalable learning. A similar trend can be seen in Figure 19 for ALR^3 and LR^3.

Finally, let us present some comparison results about the speed of convergence of ALR^n vs. LR^n as we increase n. In Figure 20, we compare the convergence for $n = 1$, $n = 2$ and $n = 3$ on the sample `Localization` dataset. It can be seen that the improvement that ALR^n provides over LR^n gets better as n becomes larger. Similar behaviour was observed for many datasets and,

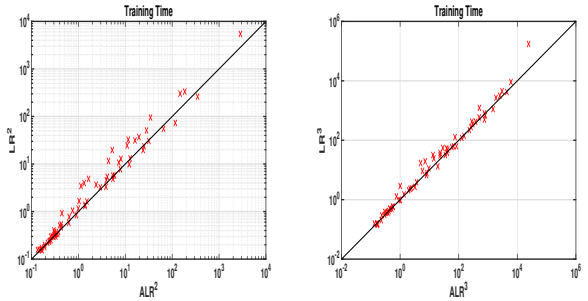

Fig. 16: Comparative scatter of Training time of ALR^2 and LR^2 (Left) and ALR^3 and LR^3 (Right) for *Little* datasets.

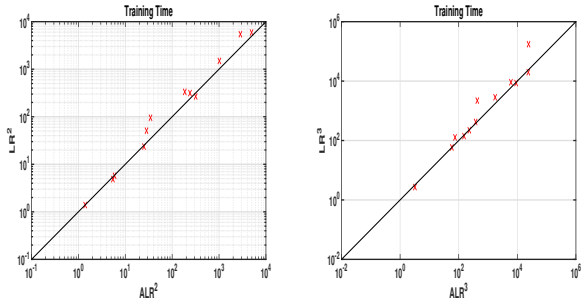

Fig. 17: Comparative scatter of Training time of ALR^2 and LR^2 (Left) and ALR^3 and LR^3 (Right) for *Big* datasets.

although studying rates of convergence is a complicated matter and is outside the scope of this work, we anticipate this phenomenon to be an interesting area for future research.

7.4 ALR^n vs. Random Forest

The two ALR^n models are compared in terms of W-D-L of 0-1 Loss, RMSE, bias and variance with Random Forest in Table 4. It can be seen that ALR^n has slightly lower bias than RF. The variance of ALR^3 is significantly higher than RF, whereas, variance does not differ significantly between ALR^2 and RF. On *Little* datasets, 0-1 Loss results of ALR^n and RF are similar. However, RF has significantly better RMSE results than ALR^n these datasets. On *Big* datasets, ALR^n has lower 0-1 Loss and RMSE on the majority of datasets.

The averaged 0-1 Loss and RMSE results are given in Figure 21. It can be seen that ALR^2, ALR^3 and RF have similar 0-1 Loss and RMSE across *Little* datasets. However, on *Big* datasets, the lower bias of ALR^n results in much lower error than RF in terms of both 0-1 Loss and RMSE. These averaged results also corroborate with the W-D-L results in Table 4, showing ALR^n to be a less biased model than RF. The comparison of training and classification

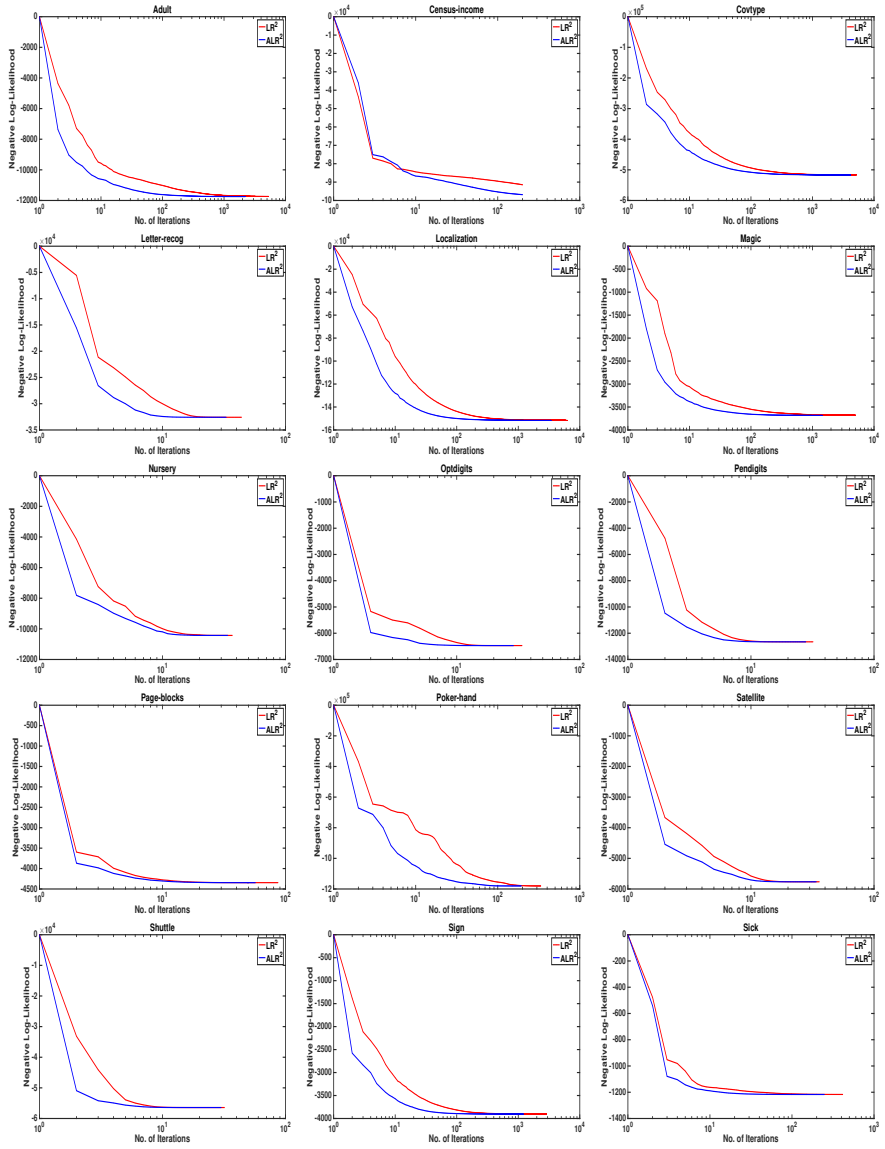

Fig. 18: Comparison of rate of convergence of ALR^2 and LR^2 on several datasets. The X-axis (No. of iterations) is on log scale. Vertical lines show the point at which the optimization is deemed to have converged.

time of ALR^n and RF is given in Figure 22. It can be seen that ALR^n requires more learning time than RF but less classification time.

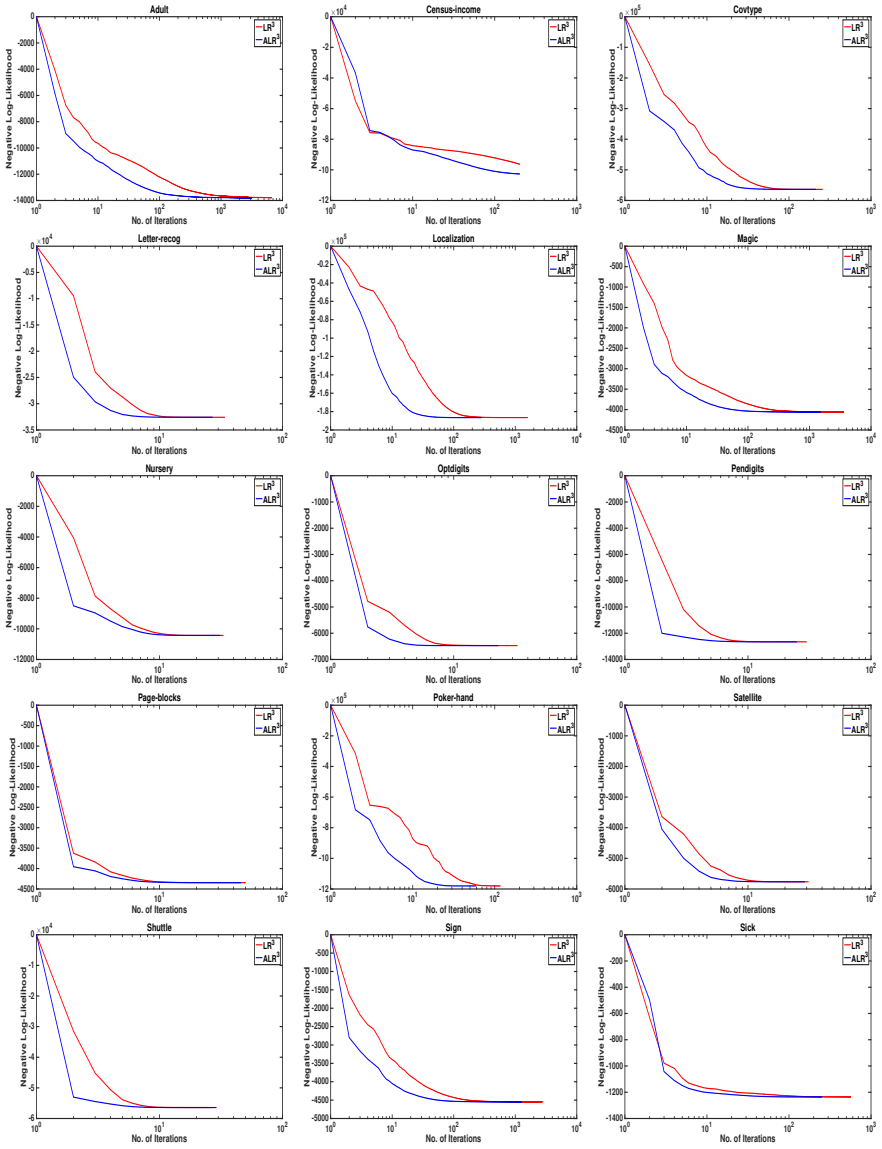

Fig. 19: Comparison of rate of convergence of ALR^3 and LR^3 on several datasets. The X-axis (No. of iterations) is on log scale. Vertical lines show the end of iterations of each curve.

8 Conclusion and Future Work

In this paper, we studied higher-order Logistic Regression (LR^n) and showed that it is a low-bias classifier that has accuracy that is highly competitive to state-of-the-art classifiers on large data. We also proposed an accelerated ver-

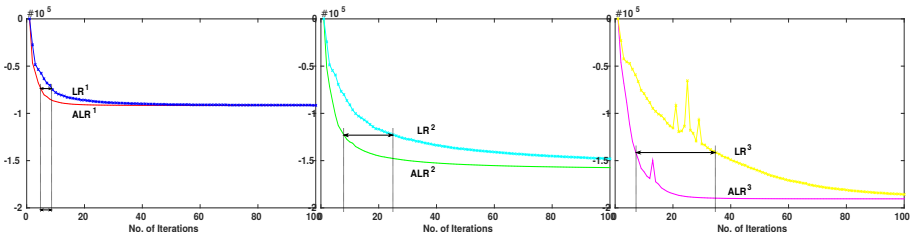

Fig. 20: Compared rates of convergence of ALR^n vs. LR^n, for $n = 1, 2, 3$ on the sample `Localization` dataset. Y-axis is the negative log-likelihood.

	ALR^2 vs. RF		**ALR^3 vs. RF**	
	W-D-L	p	W-D-L	p
		All Datasets		
Bias	39/9/28	0.221	35/9/32	0.807
Variance	25/2/49	0.556	21/3/52	$<$ **0.001**
		Little Datasets		
0-1 Loss	26/3/47	0.018	22/1/53	$<$ **0.001**
RMSE	26/0/50	0.007	25/0/51	0.003
		Big Datasets		
0-1 Loss	4/1/3	1.000	5/0/3	0.726
RMSE	4/0/4	1.273	5/0/3	0.726

Table 4: Win-Draw-Loss: ALR^2 vs. RF and ALR^3 vs RF. p is two-tail binomial sign test. Results are significant if $p \leq 0.05$.

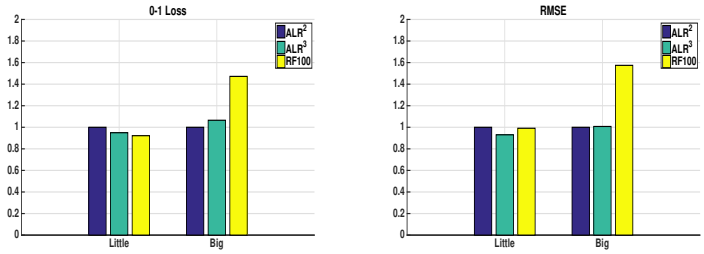

Fig. 21: Geometric mean of 0-1 Loss (Left) and RMSE (Right) performance of ALR^2, ALR^3 and RF for *Little* and *Big* datasets.

sion of higher-order Logistic Regression (ALR^n) which is based on both generative and discriminative learned parameters. To obtain the generative parameterization, we first developed AnJE, a generative counter-part of higher-order logistic regression. We showed that ALR^n and LR^n learn equivalent models, but that ALR^n is able to exploit the information gained generatively to effectively precondition the optimization process. ALR^n converges in fewer iterations, leading to its global minimum much more rapidly, resulting in faster training time. We also compared ALR^n with the equivalent AnJE and AnDE models and showed that ALR^n has lower bias than both AnJE and AnDE models. We compared ALR^n with state of the art classifier Random Forest

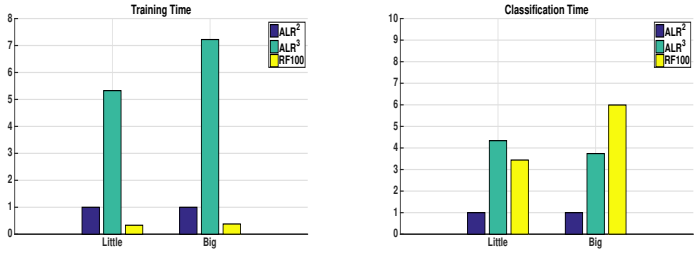

Fig. 22: Geometric average of Training Time (Left), Classification Time (Right) of ALR^2, ALR^3 and RF for *Little* and *Big* datasets.

and showed that ALR^n models are indeed lower biased than RF and on bigger datasets ALR^n often obtains lower 0-1 loss than RF.

There are a number of promising new directions for future work.

- We have shown that ALR^n is a low bias classifier that requires minimal tuning and has the ability to handle multiple classes. The obvious extension is to make it out-of-core. We argue that ALR^n is well suited for stochastic gradient descent based methods as it can converge to the global minimum very quickly.
- It may be desirable to utilize a hierarchical ALR, such that $h\text{ALR}^n = \{\text{ALR}^1 \dots \text{ALR}^n\}$, incorporating all the parameters up till order n. This may be useful for smoothing the parameters. For example, if a certain interaction does not occur in the training data, at classification time one can resort to lower values of n.
- In this work, we have constrained the values of n to two and three. Scaling-up ALR^n to higher values of n is highly desirable. One can exploit the fact that many interactions at higher values of n will not occur in the data and hence can develop sparse implementations of ALR^n models.
- Exploring other objective functions such as Mean-Squared-Error or Hinge Loss may have desirable properties and has been left as a future work.
- The preliminary version of ALR that we have developed is restricted to categorical data and hence requires that numeric data be discretized. While our results show that this is often highly competitive with Random Forests, which can use local cut-points (built-in discretization scheme), on some datasets it is not. In consequence, there is much scope for extensions to ALR^n to directly handle numeric data.

9 Code and Datasets

Code with running instructions can be download from `https://www.dropbox.com/sh/gwiah6w0w2suiaa/AAChkLCA6Iht7V6LSOVkXv5xa?dl=0`.

10 Acknowledgments

This research has been supported by the Australian Research Council (ARC) under grant DP140100087, and by the Asian Office of Aerospace Research and Development, Air Force Office of Scientific Research under contracts FA2386-15-1-4007 and FA2386-15-1-4017.

The authors would like to thank Wray Buntine, Reza Haffari and Bart Goethals for helpful discussions during the evolution of this paper. The authors also acknowledge tremendously useful comments by the reviewers that helped improve the quality of the paper.

A Results with Conjugate-Gradient Optimization

We present 0-1 Loss, RMSE, Training time and Number of Iterations results for LR^2 and ALR^2 in Figure 23 and LR^3 and ALR^3 in Figure 24 with conjugate-gradient optimization. We also compare the convergence curves in Figure 25 and 26 for LR^2 vs. ALR^2 and LR^3 vs. ALR^3 respectively.

Standard conjugate gradient's convergence criteria is used to stop convergence. Also, the maximum number of iterations were set to 10000 except for the case of `Adult` and `Covtype` datasets where maximum number of iterations are set to 200 when comparing LR^3 and ALR^3. It can be see that both LR^n and ALR^n have a similar spread of 0-1 Loss and RMSE, but ALR^3 has an advantage of better training time and converges not only in fewer iterations but asymptote to global minimum much more quickly than LR^n.

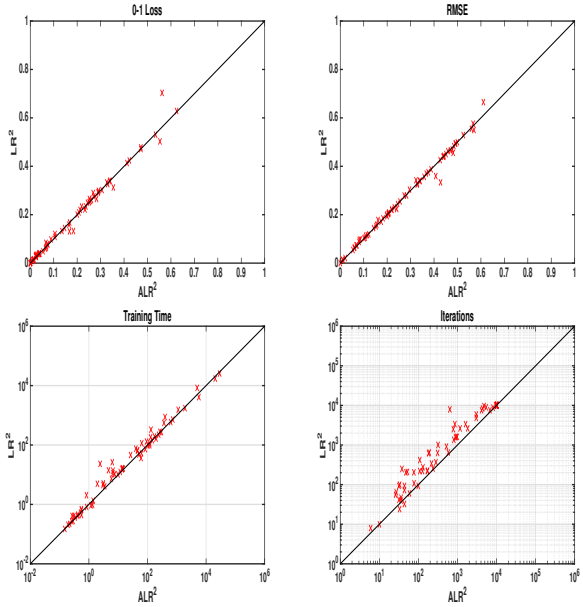

Fig. 23: Geometric mean of 0-1 Loss (Top Left), RMSE (Top Right), Training Time (Bottom Left) and No. of Iterations (Bottom Right) performance of ALR^2 and LR^2 (trained with Conjugate Gradient) on all datasets.

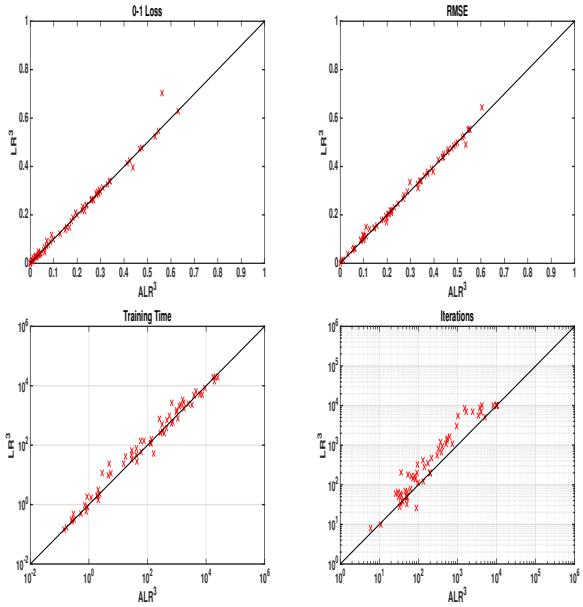

Fig. 24: Geometric mean of 0-1 Loss (Top Left), RMSE (Top Right), Training Time (Bottom Left) and No. of Iterations (Bottom Right) performance of ALR^3 and LR^3 (trained with Conjugate Gradient) on all datasets.

B On Initialization of weights in ALR^n

In Section 5.1, we derived a suitable weighting for AnJE that was equal to $\gamma^n = \frac{(n-1)!(a-n)!}{(a-1)!}$. We discussed how this weight motivates the introduction of a different weight for each sub-model and for each feature sub-set. In this section, we will study if this weight (i.e., γ^n) can be of any help in finding the weights in ALR^n by exploring the following two possibilities:

1. Scale the weights learned by ALR^n by γ^n at every iteration of optimization.
2. Initialize ALR^n weights by γ^n and optimize.

Scaling ALR^n weights with γ^n leads to an objective function of the form:

$$\text{P}_{\text{ALR (J)}}(y\,|\,\mathbf{x}) = \exp\Big(w_y \log \text{P}(y) + \sum_{\alpha \in \binom{\mathcal{A}}{n}} \gamma^n w_{y,\alpha,x_\alpha} \log \text{P}(x_\alpha\,|\,y) - \log \sum_{c \in \Omega_Y} \exp\Big(w_c \log \text{P}(c) + \sum_{\alpha^* \in \binom{\mathcal{A}}{n}} \gamma^n w_{c,\alpha^*,x_{\alpha^*}} \log \text{P}(x_{\alpha^*}\,|\,c)\Big)\Big). \quad (22)$$

Note, we have denoted this version of ALR by ALR (J). Also, γ^n will appear in the gradient of above objective function.

The comparative scatter plots of 0-1 Loss, RMSE, training time and number of iterations to convergence for each algorithm (i.e., ALR and ALR (J)) are shown in Figure 27 for $n = 2$ and in Figure 28 for $n = 3$. It can be seen that with a similar 0-1 Loss and RMSE profile, training time and number of iterations of ALR (J) is worse than that of ALR. Therefore, plugging γ^n in to the ALR^n objective function does not result in faster convergence of ALR.

The second option is to initialize w_{y,α,x_α} with γ^n instead of 0 or 1 (that is typical of gradient-based optimization techniques). We will denote this version of ALR as ALR (init). The comparative scatter plots of 0-1 Loss, RMSE, training time and number of iterations

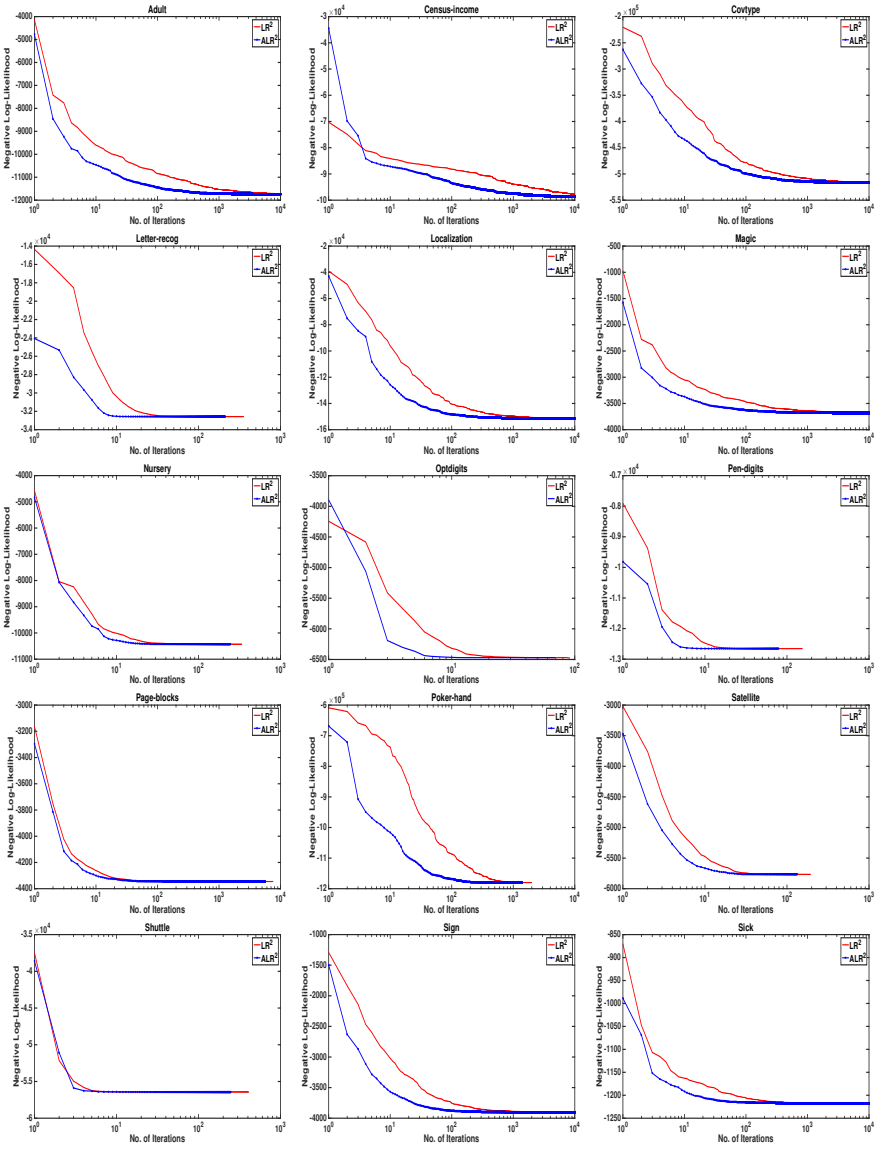

Fig. 25: Comparison of rate of convergence of LR^2 and ALR^2 (trained with Conjugate Gradient) on several datasets. The X-axis (No. of iterations) is on log scale.

it takes each algorithm (i.e., ALR and ALR (init)) are shown in Figure 29 for $n = 2$ and in Figure 30 for $n = 3$ respectively. It can be seen that both $n = 2$ and $n = 3$ share similar 0-1 Loss and RMSE profiles. However, training time and number of iterations are slightly better for ALR (init) as compared to ALR. This is intuitive, as γ^n guides the weights to a slightly better starting place in the optimization search space that saves some iterations to reach the global minimum. However, the speed-up is not significant.

We compare the convergence curves of ALR, ALR (J) and ALR (init) on six sample datasets (`Adult, Connect-4, Localization, Covtype, Census-income, Poker-hand`)

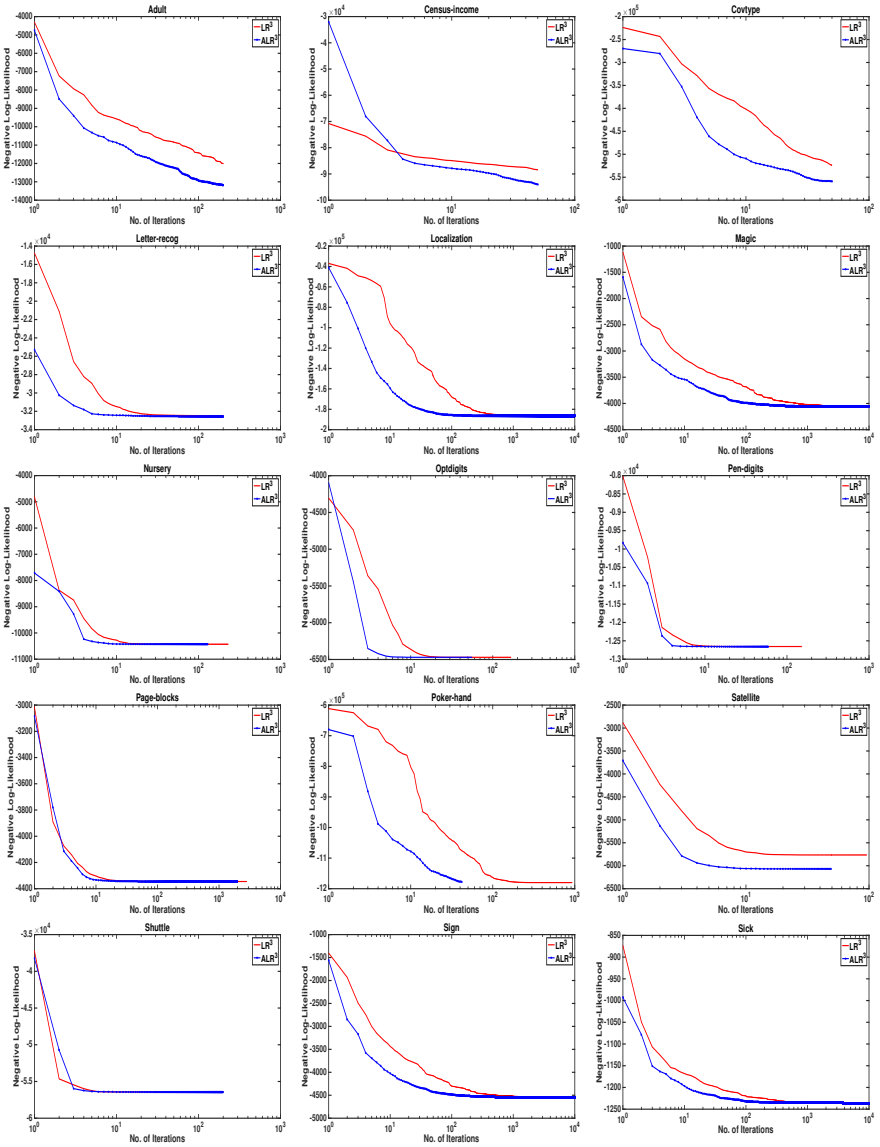

Fig. 26: Comparison of rate of convergence of LR^3 and ALR^3 (trained with Conjugate Gradient) on several datasets. The X-axis (No. of iterations) is on log scale. Note, that on `Adult` and `Covtype` dataset, maximum number of iterations are set 200.

in Figures 31 and 32 for $n = 2$ and $n = 3$ respectively. It can be seen that ALR (init) has the best convergence rate. It starts from the better (lower) spot (except for `census-income`) and then asymptotes in line with the ALR curve. ALR (J), on the hand, has an adverse effect on the convergence of ALR.

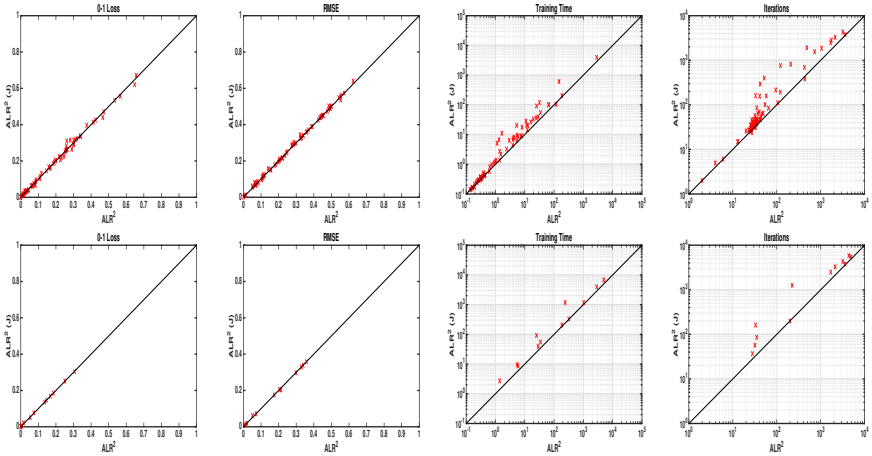

Fig. 27: (Top Row) Comparative scatter of 0-1 Loss, RMSE, Training Time and number of Iterations of of ALR^2 and ALR^2 (J) for *Little* datasets. (Bottom Row) Comparative scatter of 0-1 Loss, RMSE, Training Time and No. of Iterations of of ALR^2 and ALR^2 (J) for *Big* datasets.

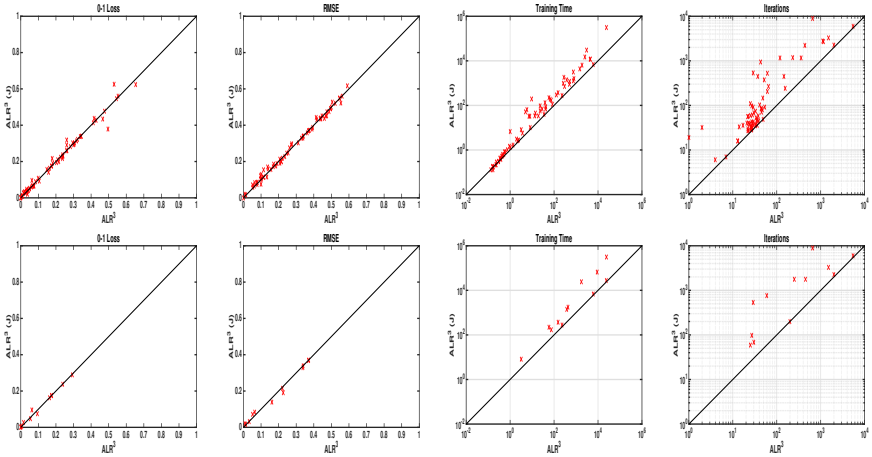

Fig. 28: (Top Row) Comparative scatter of 0-1 Loss, RMSE, Training Time and number of Iterations of of ALR^3 and ALR^3 (J) for *Little* datasets. (Bottom Row) Comparative scatter of 0-1 Loss, RMSE, Training Time and No. of Iterations of of ALR^3 and ALR^3 (J) for *Big* datasets.

References

1. Bishop, C.: Pattern Recognition and Machine Learning. Springer (2006)
2. Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge Unversity Press (2008)
3. Brain, D., Webb, G.I.: The need for low bias algorithms in classification learning from small data sets. In: PKDD, pp. 62–73 (2002)
4. Breiman, L.: Random forests. Machine Learning **45**, 5–32 (2001)
5. Byrd, R., Lu, P., Nocedal, J.: A limited memory algorithm for bound constrained optimization. SIAM Journal on Scientific and Statistical Computing **16**(5), 1190–1208 (1995)

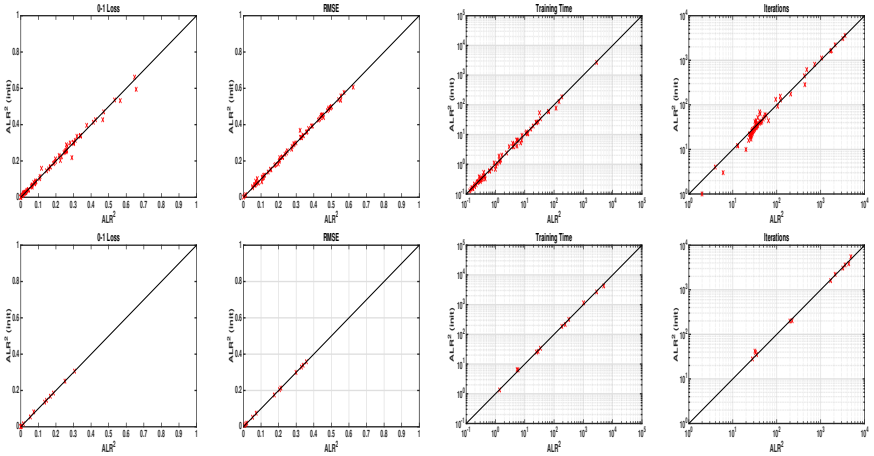

Fig. 29: (Top Row) Comparative scatter of 0-1 Loss, RMSE, Training Time and number of Iterations of of ALR^2 and ALR^2 (init) for *Little* datasets. (Bottom Row) Comparative scatter of 0-1 Loss, RMSE, Training Time and No. of Iterations of of ALR^2 and ALR^2 (init) for *Big* datasets.

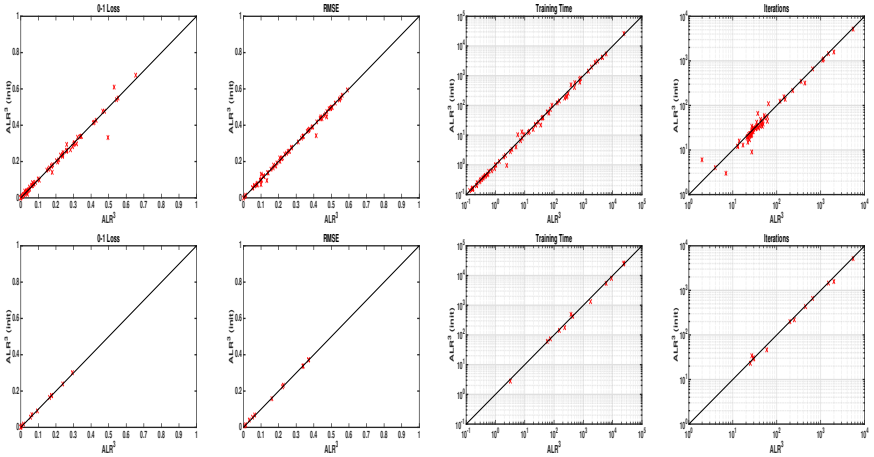

Fig. 30: (Top Row) Comparative scatter of 0-1 Loss, RMSE, Training Time and number of Iterations of of ALR^3 and ALR^3 (init) for *Little* datasets. (Bottom Row) Comparative scatter of 0-1 Loss, RMSE, Training Time and No. of Iterations of of ALR^3 and ALR^3 (init) for *Big* datasets.

6. Fayyad, U.M., Irani, K.B.: On the handling of continuous-valued attributes in decision tree generation. Machine Learning **8**(1), 87–102 (1992)
7. Firth, D.: Bias reduction of maximum likelihood estimates. Biometrika **80**, 27–38 (1993)
8. Frank, A., Asuncion, A.: UCI machine learning repository (2010). URL `http://archive.ics.uci.edu/ml`
9. Ganz, J., Reinsel, D.: The Digital Universe Study (2012)
10. Genkin, A., Lewis, D., Madigan, M.: Large-scale bayesian logistic regression for text categorization. Technometrics **49**, 291–304 (2012)
11. Greiner, R., Su, X., Shen, B., Zhou, W.: Structural extensions to logistic regression: Discriminative parameter learning of belief net classifiers. Machine Learning Journal

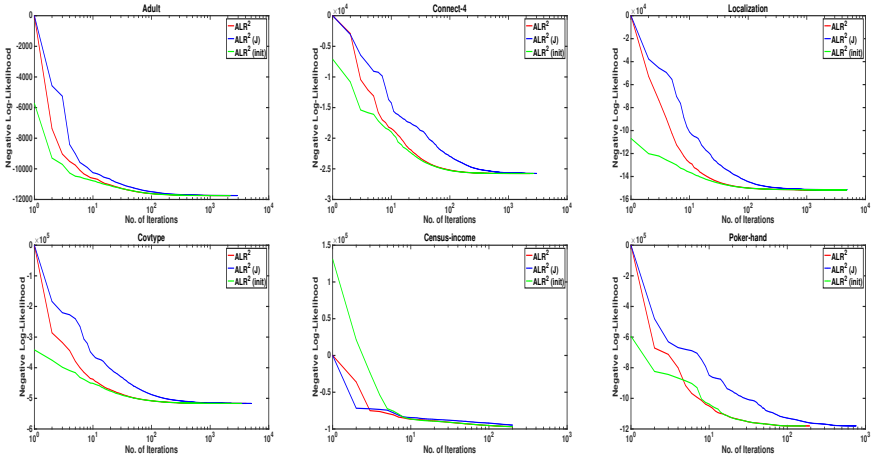

Fig. 31: Comparative scatter of 0-1 Loss, RMSE, Training Time and number of Iterations of of ALR^2 and ALR^2 (init) for *Little* datasets.

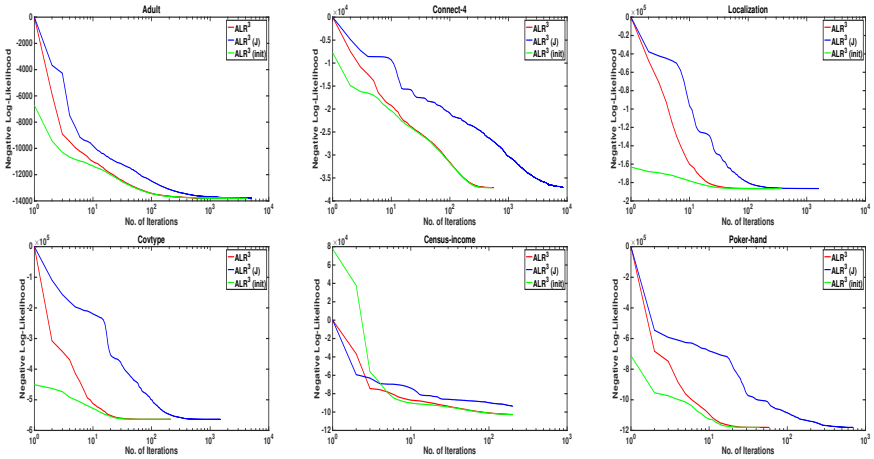

Fig. 32: Comparative scatter of 0-1 Loss, RMSE, Training Time and number of Iterations of of ALR^3 and ALR^3 (init) for *Little* datasets.

(2005)
12. Greiner, R., Zhou, W.: Structural extension to logistic regression: Discriminative parameter learning of belief net classifiers. In: AAAI (2002)
13. Hauck, W., Anderson, S., Marcus, S.: Should we adjust for covariates in nonlinear regression analysis of randomised trials? Controlled Clinical Trials **19**, 249–256 (1998)
14. Inc., D.: Statistics: Methods and applications. `http://documents.software.dell.com/statistics/current/textbook` (2013)
15. Kohavi, R., Wolpert, D.: Bias plus variance decomposition for zero-one loss functions. In: ICML, pp. 275–283 (1996)
16. Langford, J., Li, L., Strehl, A.: Vowpal wabbit online learning project (2007)
17. Lauritzen, S.: Graphical models. Oxford University Press. (1996)
18. Lin, X., Wahba, G., Xiang, D., Gao, F., Klein, R., B., K.: Smoothing spline anova models for large data sets with bernoulli observations and the randomized gacv. Tech.

rep., Technical Report 998, Department of Statistics, University of Wisconsin, Madison WI (1998)
19. Lint, J.H., M, W.R.: A Course in Combinatorics. Cambridge University Press. (1992)
20. Liu, H., Motoda, H.: Feature extraction, construction and selection: A data mining perspective. Springer Science & Business Media (1998)
21. Martinez A. Chen, S., Webb, G.I., Zaidi, N.A.: Scalable learning of Bayesian network classifiers. Journal of Machine Learning Research (2015)
22. Mitchell, T.M.: The need for biases in learning generalizations. Technical Report CBM-TR-117, Rutgers University, Department of Computer Science, New Brunswick, NJ (1980)
23. Neuhaus, J., Jewell, N.: A geometric approach to assess bias due to omitted covariates in generalized linear models. Biometrika **80**, 807–815 (1993)
24. Pazzani, M.J.: Constructive induction of cartesian product attributes. In: In Proceedings of the Information, Statistics and Induction in Science Conference (ISIS96, pp. 66–77 (1996)
25. Pernkopf, F., Bilmes, J.: Discriminative versus generative parameter and structure learning of Bayesian network classifiers. In: ICML (2005)
26. Pernkopf, F., Wohlmayr, M.: On discriminative parameter learning of Bayesian network classifiers. In: ECML PKDD (2009)
27. Roos, T., Wettig, H., Grünwald, P., Myllymäki, P., Tirri, H.: On discriminative Bayesian network classifiers and logistic regression. Machine Learning **59**(3), 267–296 (2005)
28. Sahami, M.: Learning limited dependence Bayesian classifiers. In: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, pp. 334–338. AAAI Press, Menlo Park, CA (1996)
29. Smola, A., Scho?lkopf, B.: Sparse greedy matrix approximation for machine learning. In: ICML (2000)
30. Sonnenburg, S., Franc, V.: Coffin: A computational framework for linear svms. In: ICML (2010)
31. Steinwart, I.: Sparseness of support vector machines - some asymptotically sharp bounds. In: NIPS (2004)
32. Stinson, D.R.: Combinatorial Designs: Constructions and Analysis. Springer (2003)
33. Su, J., Zhang, H., Ling, C., Matwin, S.: Discriminative parameter learning for Bayesian networks. In: ICML (2008)
34. Szilard, N., Jonasson, J., Genell, A., Steineck, G.: Bias in odds ratios by logistic regression modelling and sample size. BMC Medical Research Methodology **Volume 9/1**, 1–5 (2009)
35. Webb, G.I.: Multiboosting: A technique for combining boosting and wagging. Machine Learning **40**(2), 159–196 (2000)
36. Webb, G.I., Boughton, J., Wang, Z.: Not so naive Bayes: Averaged one-dependence estimators. Machine Learning **58**(1), 5–24 (2005)
37. Webb, G.I., Boughton, J., Zheng, F., Ting, K.M., Salem, H.: Learning by extrapolation from marginal to full-multivariate probability distributions: decreasingly naive Bayesian classification. Machine Learning pp. 1–40 (2011). URL `http://dx.doi.org/10.1007/s10994-011-5263-6`. 10.1007/s10994-011-5263-6
38. Williams, C., Seeger, M.: Using the nystrom method to speed up kernel machines. In: NIPS (2001)
39. Zaidi, N.A., Carman, M.J., Cerquides, J., Webb, G.I.: Naive-Bayes inspired effective pre-conditioners for speeding-up logistic regression. In: IEEE International Conference on Data Mining, pp. 1097–1102 (2014)
40. Zaidi, N.A., Cerquides, J., Carman, M.J., Webb, G.I.: Alleviating naive Bayes attribute independence assumption by attribute weighting. Journal of Machine Learning Research **14**, 1947–1988 (2013)
41. Zaidi, N.A., Webb, G.I.: Fast and efficient single pass Bayesian learning. In: Advances in Knowledge Discovery and Data Mining, pp. 149–160 (2012)
42. Zhu, J., Hastie, T.: Kernel logistic regression and the import vector machine. In: NIPS, pp. 1081–1088 (2001)