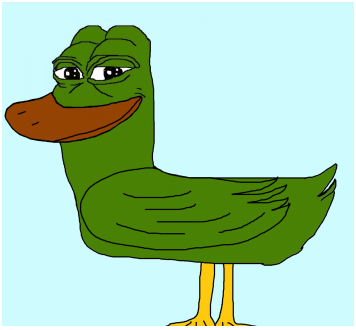


Mitigating Covariate Shift for Low Dimensional Machine Learning Problems via Lattice Based Models

Bachelor's Thesis submitted

to

Prof. Dr. X Man

Infinite-Jest-Universität zu Berlin

School of Joy and Fun

Chair of Entertainment Systems

by

jvosten

654321

in partial fulfillment for acquiring

the degree of

Bachelor of Science (B.Sc.)

in Economics

Berlin, February 29*th*, 2021

Contents

1 Introduction

In recent years *data* has become an important topic for the broader public, exceeding its natural habitat of science and computer programming. Some proclaimed it to be the oil of the $21th$ century. A major reason for the rush lies in the availability of data and the development of new technologies and methods, which enable us to design algorithms that learn from data. This emerging scientific field is known as Machine Learning (ML), some even see the advent of a new scientific age:

> The new availability of huge amounts of data, along with the statistical tools to crunch these numbers, offers a whole new way of understanding the world. Correlation supersedes causation, and science can advance even without coherent models, unified theories, or really any mechanistic explanation at all.[1]

To temper the above quoted author's excitement, we could simply ask about the Problem of Induction; the question of whether it is possible to infer a generally valid law based on single observations. Presuming we go bird watching and observe a 100 swans, all of them with white plumage, we could then draw the following two inductive conclusions:

(C1) All swans observed so far were white. Therefore all swans are white. $F(a_1), \ldots, F(a_n) \Rightarrow \forall x : F(x)$.

(C2) All swans observed so far were white. Therefore the next observed swan will be white. $F(a_1), \ldots, F(a_n) \Rightarrow F(a_{n+1})$.

This poses two questions regarding justification: how do we justitfy generalizing about some object x's features F based on n observations of a particular instance $F(a_1), \ldots, F(a_n)$; and how do we justify the assumption, that future observations a_{n+1} are going to fall under F, as the past ones. The *Problem of Induction* forms one of the fundamental problems in the domain of Epistemology and Theory of Machine Learning Sterkenburg and Grünwald (2020). In bringing this up, we remark that (statistical) learning from data faces the same problems as any field utilizing inductive inferrence. Regarding ML we could then alter $C1$ and $C2$ in the following way

(C3) All observations x_i for feature x fall within the range of $[a, b]$. Therefore the model learns $y \in [a, b]$.

(C4) All observations x_i for feature x fall within the range of $[a, b]$. Therefore the model predicts $y \in [a, b]$ for x_{n+1}.

[1] `https://www.wired.com/2008/06/pb-theory/`

One could insist that the reasoning of $C3$ and $C4$ does not seem problematic and refers us to all those well-functioning ML models and applications. But, especially when it comes to real world ML applications, the following problem is very common: the data $x_{ts} \in [c, d]$, on which the model $f(x)$ will be tested on in the end by the user, is not strictly (or not at all) represented by the data $x_{tr} \in [a, b]$ that was used for training the underlying model. This kind of violation of $C3$ and $C4$ is called Dataset Shift (Moreno-Torres et al., 2012). More technically speaking, dataset shift appears when training and test joint distributions are different (Quionero-Candela et al., 2009). Moreno-Torres et al. identify three main types of dataset shift, covering $(i.)$ a shift in the independent variables (*Covariate shift*), $(ii.)$ in the target variable (*Prior probability shift*) or $(iii.)$ in the relationship between the two (*Concept shift*) .

In this paper we will focus on the first type, covariate shift. In this case the values of the covariates X causally determine the class label Y. A shift then occurs, if the distribution of one or more of the covariates X_{ts} in the test data significantly changes compared to X_{tr} in the training data. Our goal then is to employ a monotonic lattice based model to mitigate the effect of covariate shift. This kind of model learns flexible monotonic functions by using calibrated interpolated look-up tables, the lattice (Gupta et al., 2016). We can think of it as looking up a value x for $\sin(x)$ in a textbook. The monotonicity constraint for individual features allows us to model prior knowledge. For instance, assume we want to estimate the risk of lung cancer for a patient. In the patients record we might find a covariate 'cigarettes smoked per day'. We expect the risk of lung cancer never to decrease as the amount of cigarettes smoked per day increases. This prior domain knowledge could nout be reflected in a model that learns from a small and noisy data set (see also the example of Fig. 1 and 2).

There is one major restriction to the lattice approach, though. It is only suitable for low dimensional ML problems, because the weights of the function are computed in $\mathcal{O}(2^D)$ operations. For the experiment in this paper we will use data from the domain of credit scoring, as the information on potential customers, characterized by the covariates, determines the credit score Y. Credit scroing is very suitable for our endeavour, as it can be considered a low dimension setting (Lessmann et al., 2015, Tab. 1) and covariate shift can be a problematic issue (see 3.1). The goal of the experiment is to show that a TensorFlow Lattice (TFL) calibrated linear model (GAM) does not perform worse than a comparison classifier, in this case a Random Forrest model.

The paper is organized as follows. We begin with an overview of relatedwork in covariate

shift adaption. Then we outline the theoretical conception of covariate shift, lattice regression and extionsions of the lattice model. Sec. 4 describes the experiment set-up, followed by a presentation of the results. Finally, we draw a conclusion in Section 6.

2 Related Work

Since mitigation of covariate shift is the main topic of this paper, the literature review is going to be focused on this aspect. For a review of similar learning methods as the montonic calibrated lattice applied here, see (Gupta et al., 2016). For a review of classifier selection for credit scoring, see Baesens et al. (2003) respectively (Lessmann et al., 2015).

We furthermore restrict our literature review to methods dealing with what (Storkey, 2009) calls 'Simple covariate shift' (p. 7) in the pardigm of unsupervised learning; that is if the complete data set shift is characterized by what we will define as covariate shift in Sec. 3.1: $P_{tr}(x) \neq P_{tst}(x)$, everything else stays the same. Exceeding this rather simple case of data set shift there exists a broad literature on domain and method specific ways to mitigate cases in which covariate shift is only one aspect of bias; see for instance (Kato et al., 2020) or (Johansson et al., 2018).

As we will see in Sec. 3.1 there is no general definition for covariate shift, neither is there even a common naming convention for the general problem of data set shift (Moreno-Torres et al., 2012). This makes it complicated to compile a review of the literature on the topic, as some work could have been overseen simply because of the aforementioned. Apart from this we can identify two superordinated approaches for covariate shift correction: (*i.*) importance weighting and (*ii.*) error bounds for domain adaption (statistical learning bounds).

Importance Weighting

Given we know the probabilites of finding x in the train and test set, $P_{tr}(x)$ and $P_{tst}(x)$, (Shimodaira, 2000) showed that under covariate shift the log-likelihood estimation becomes asymptotically optimal, if each instance is reweighted with

$$w(x) = \frac{P_{tst}(x)}{P_{tr}(x)}.$$

An individual training point x_i receives a higher weight, if its probability of appearing in the test set is high. Multiplying the resulting weights with the initial input is supposed to result in a more accurate prediction. (Nair et al., 2019) give an overview on the most important methods for estimating $w(x)$ following the above sketched *Importance Weighting*:

Discriminative Learning (Bickel et al., 2009) propose to use a probabilistic classifier for estimating $w(x)$. The probabilites are estimated by a discriminative algorithm; they can furthermore be utilized as a measure to decide upon the existence of covariate shift for the data set of concern. More efficient versions of this method have been introduced by (Sugiyama, 2010) (least-squares probabilistic classifier) and (Yamada et al., 2011) (importance weighted LSPC).

Kernel Mean Matching Here training and test data are mapped to reproducing kernel Hilbert spaces (RKHS) and distances between mappings are measured. The algorithm then reweights the training data in a way that their RKHS mapping corresponds to that of the test data (Gretton et al., 2009), (Huang et al., 2006). This approach has the advtange that no estimation of the densities is required. It got extended by (Kanamori et al., 2009b) and (Miao et al., 2015).

KLIEP (*Kullback Leibler Importance Estimation Procedure*) Model Selection for the above KMM method may be biased. (Sugiyama et al., 2008) therefore propose a method that directly estimates $w(x)$ without density estimation and that contains a proper model selection, by minimizing the Kullback-Leibler divergence from $P_{tst}(x)$ to its estimate $\hat{P}_{tst}(x) = \hat{w}(x)P_{tr}(x)$.

(u)LSIF *(Unconstrained) Least Squares Importance Fitting (uLSIF)* Similar method as KLIEP that uses squared loss to model the importance estimation (Kanamori et al., 2009a). It is supposed to be more efficient then KLIEP. uLSIF is a numerically stable version of LSIF.

In addition, there are further approaches to the problem, as *Asymptotic Bayesian Generalization Error* (Yamazaki et al., 2007), *Importance Weighted Cross Validation* (Sugiyama et al., 2007) or *Adversarial Search* (Globerson et al., 2009).

Statistical Learning

(Ben-David, 2009) provides a setting of domain adaption for which he derrives error bounds. These bounds forego assumptions on the similarity between the domain of train and test data. Ben-David proposes two different algorithms, which he calls 'adaptive inductive transfer learning algorithms', see (Ben-David and Schuller, 2003) and (Ben-David et al., 2007).

In addition, an exhaustive overview on covariate shift adaption can be found in (Sugiyama and Kawanabe, 2012).

3 Theoretical Foundations

In this section we provide a brief introduction into covariate shift and illustrate, how it can be related to credit scoring. In the subsections below, we will then explore the lattice regression framework and its monotonicity and calibration extensions.

3.1 Covariate Shift

In supervised machine learning, it is often assumed that the test and training data follow the same distributions, or that it does not matter if their domains do not match. In real-world applications this mismatch can lead to models performing well in a certain context, but failing in other contexts, where, compared to the training context, the data shifted. Data set shift therefore describes the effects of the violation of the above mentioned assumption. In the literature there are several approaches to systemize data set shift: (Storkey, 2009) describes causes of data set shift, (Moreno-Torres et al., 2012) systemize four different effect types of shift (the three types stated in Sec. 1 plus 'other types') and (Kull and Flach, 2014) refine the aforementioned approaches by extending the categorization and by supplementing it with a formal notation using graphical models. For our purpose it will be sufficient to refer to the simple distinction of shifts as described by Moreno-Torres et al., as the specifications of Kull and Flach concern relatively particular cases.

(Moreno-Torres et al., 2012) model dataset shift as a change in the joint probability distribution $P(y, x)$ of the covariates X and the labels Y:

> ***Dataset shift*** *appears when training and test joint distributions are different. That is, when* $P_{tr}(y, x) \neq P_{tst}(y, x)$.

The authors give a further specification of the joint distribution $P(y, x)$ to take into consideration the causal relationship between covariates and class labels. The kind of shift that is possible depends on the structure of the problem, leading to the distinction of two cases: for $X \rightarrow Y$ problems values of covariates causally determine the class label; for $Y \rightarrow X$ problems class labels determine the values of covariates. For the former the joint probability distribution $P_{tr}(y, x) = P_{tst}(y, x)$ can then be written as $P(y|x)P(x)$, for the later as $P(x|y)P(x)$.

From the above Moreno-Torres et al. obtain a formal definition of covariate shift:

> ***Covariate shift*** *appears only in* $X \rightarrow Y$ *problems, and is defined as the case where* $P_{tr}(y|x) = P_{tst}(y|x)$ *and* $P_{tr}(x) \neq P_{tst}(x)$.

Here only the distributions of x for test and train sets change, the functional relationship of the model stays the same. For a mathematical more rigorous description of the entire problem setting see (Sugiyama and Kawanabe, 2012). The definitions of the other types of shift are summarized in Tab. 1.

	$X \to Y$ Problem		$Y \to X$ Problem	
	$P_{tr}(y\|x) = P_{tst}(y\|x)$	$P_{tr}(x) = P_{tst}(x)$	$P_{tr}(x\|y) = P_{tst}(x\|y)$	$P_{tr}(y) = P_{tst}(y)$
Covariate Shift	True	False	–	–
Prior Probability Shift	–	–	True	False
Concept Shift	False	True	False	True
Other types	False	False	False	False

Table 1: The four types of dataset shift displayed in form of a logical truth table for both kinds of problem. Each depicted case leads to $P_{tr}(y, x) \neq P_{tst}(y, x)$.

To illustrate the relationship between covariate shift and credit scoring we draw on an example from (Wang and Gupta, 2020, Sec. 8.2). Here, a credit scoring model is employed based on the Taiwanese UCI Credit Card data set, which we also will use for our experiment in Sec. 4. The authors remark a potenital bias for the variable 'Repayment status (April)', as the data appears to be very noisy for cases with *overdue* > 3 (see Appendix Fig. 10). In a simplified two-feature test case (with covariates 'Marital status' and 'Repayment status (April)') their model then yields the output visualized in Fig. 1:

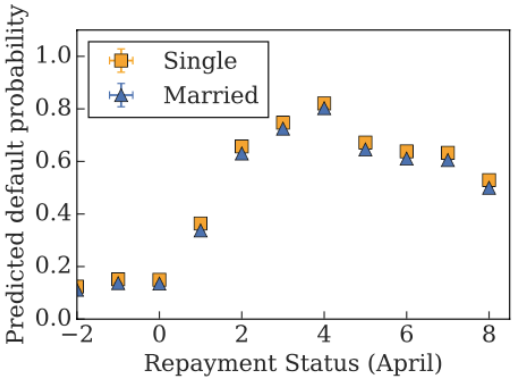


Figure 1: Baseline model

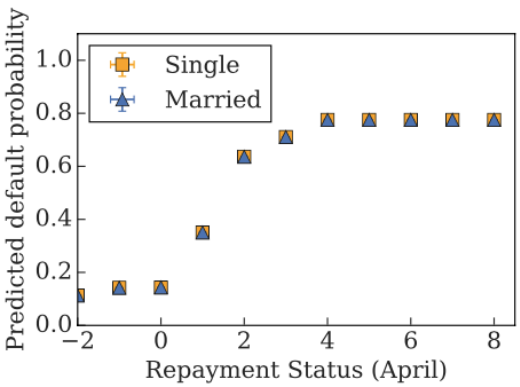


Figure 2: Lattice model

Fig. 1 shows that customers who are 5 or 6 month overdue, are assigned a lower score than those, who are only 2 months overdue. This means, borrowers are penalized by the score for paying back in time. The underlying bias most likely results from the training data

only containing 122 cases of *overdue* > 3, leaving those cases underrepresented in a training set of 30.000 observations. We can then model this example as a case of covariate shift.

Assume we now want to apply the classifier in a new context. In our new test set there is a significantly higher amount of observations with *overdue* > 3 for 'April repayment', let it be 5% instead of 0.4%. We then have $P_{tr}(\text{April repayment}) \neq P_{tst}(\text{April repayment})$, potentially penalizing those customers who are only slightly overdue. All other variables fixed, those customers obtain a worse credit score then heavy overdue customers. The functional relationship $P(y|x)$ remains unchanged, but the input distribution $p(x)$ differs.

To mitigate the bias of the estimator, Wang and Gupta supplement their lattice model with a *mononicity shape constraint*. In Fig. 2 we see a more consistent behaviour of the estimator, as the shape constraint permits to integrate prior knowledge into the model. For their experiment the authors used a nonlinear generalized additive model, respectively a TFL calibrated linear model. In the following subsections we explore the concept behind the lattice modeling approach.

3.2 Lattice Regression

Lattice regression is the name for a regression method, in which the estimated function is represented as a rectangular grid of function values spanning the input domain. As in a textbook lookup table for the z-values of standard normal distribution, any needed value can then be linearly interpolated from the lattice. In developing supervised machine learning applications we usually start from scratch fitting a function $f(x)$ to the available training data by using some regression algorithm, with the data consisting of randomly sampled pairs $\{(x_i, y_i)\}$. In a second step $f(x)$ can then be evaluated on a lattice to generate a look up table. There is a disadvantage to this two-step procedure, though, as the effect of interpolation from the lattice nodes (on the error of the training samples) is not taken into account, respectively the empirical risk function does not consider the interpolation step. To solve this issue, (Garcia and Gupta, 2009, pp. 1)

> propose a solution that we term lattice regression, that jointly estimates lattice outputs by minimizing the regularized interpolation error on the training data. [...] The key to this estimation is that the linear interpolation operation can be directly inverted to solve for the node outputs that minimize the squared error of the training data.

The major constraint of this approach is the exponential scaling of the interpolation operation with $\mathcal{O}(2^D)$, as each evaluation of $f(x)$ calls for the interpolation of the 2^d lattice nodes.

Therefore lattice regression rather suits low-dimensional problems.

In the following subsections we will provide an overview on lattice regression. At first we take a look at the regression objective; then we will describe the interpolation function for the function output and last we present different regulizers for the estimation function.

3.2.1 Regression Objective

In regression we commonly try to estimate a function $\hat{f} : D \to \mathbb{R}$ within some class of functions $\mathcal{F}$ to minimize the empirical risk $\sum \ell(y, \hat{f}(x))$ for some loss funtion $\ell(\cdot)$. In lattice regression the class of admissible functions $\mathcal{F}$ is then restricted to the following: given the bounded input space $D \subset \mathbb{R}^d$ (with $d = 1, \ldots, D$ being the amount of covariates) from which we draw the training data with inputs $\{x_i \in D\}_{i=1:n}$ and outputs $\{y_i \in \mathbb{R}\}_{i=1:n}$. We then assume a lattice spanning the domain D, consisting of $M = \prod_d M_d = M_1 \times M_2 \times \ldots M_D$ vertices, whereby $M_d \in \mathbb{N}$ is to be considered a hyperparameter which determines the number of vertices in the lattice for the dth covariate. Each individual vertex of M then consists in an input-output pair $(v_j \in \mathbb{R}^d, \theta \in \mathbb{R}^M)$, with the inputs v_j forming a lattice that spans the hyper-rectangle $\mathcal{M} \triangleq [0, M_1 - 1] \times [0, M_2 - 1] \times \ldots [0, M_D - 1]$ and the outputs θ which are the learned lattice values of $f(x)$. For reasons of notational simplicity we follow (Gupta et al., 2016) and assume a 2^D lattice such that $x \in [0, 1]^D$; furthermore we follow (Garcia et al., 2012) assuming a domain transformation to obtain a lattice with one corner placed at the origin and regularly spaced vertices. Fig. 3 shows such a 2^D lattice with $D = 2$. Then the class of admissible functions $\mathcal{F}$ is restricted by those functions 'that can be implemented by linearly interpolating a rectangular lattice' (Garcia et al., 2012, Sec. 2). Finally any function in $\mathcal{F}$ is fully defined by the output set $\{\theta\}$, given lattice inputs $\{v_j\}$ and a linear interpolation method for calculating the interpolation weights $\phi(x)$.

For linear interpolation of x_i the interpolation weights $\phi(x_i)$ are calculated. The sum of these weight vectors can be written as a uniquely determined matrix $\phi(x) \in [0, 1]^{n \times M}$, as they depend on the lattice vertices $\{v_j\}$ and the input data $\{x_i\}$. To be more precise, the set of linear interpolation weights $\phi(x)$ is the solution to the following system of equations:

$$\sum_{k=0}^{2^D} \phi_k(x_i) v_k = x_i \text{ and } \sum_{k=0}^{2^D} \phi_k(x_i). \tag{1}$$

As we obtain θ through $\phi(x)$, we can finally compute $\hat{f}(x_i) = \theta^T \phi(x_i)$. The goal of lattice regression is then to minimize the interpolation error on the training data, ergo to minimize

the empirical risk function by solving for lattice output θ:

$$\begin{aligned}\hat{\theta} &= arg \min_{\theta \in \mathbb{R}^M} \sum_{i=1}^{n} \ell(y_i, \hat{f}(x_i)) \\ &= arg \min_{\theta \in \mathbb{R}^M} \sum_{i=1}^{n} \ell(y_i, \theta^T \phi(x_i)).\end{aligned} \tag{2}$$

Assuming squared loss $\ell(y, z) = (y - z)^2$ for the loss funcion ℓ in the empirical risk alters Eq. 2 into the following equation:

$$\hat{\theta} = arg \min_{\theta \in \mathbb{R}^M} \left\| y - \theta^T \phi(x) \right\|_2^2 . \tag{3}$$

Eq. 3 yields us the objective function of a linear regression model which finds a solution in $(\phi(x)^T \phi(x)^{-1}) \phi(x)^T y$. If Eq. 3 is underdetermined, there is no solution. (Garcia et al., 2012) remark this to be the case often. There are for instance cases, in which $\phi(x)$ will not be invertible, leaving Eq. 3 with an infinite set of minimizers. To prevent this problem and further to reduce overfitting to the training data, a graph regulizer $R(\theta)$ is added to the regression objective. Before we examine potential regulizers, we take a closer look at the interpolation function $\phi(x_i)$ in the following subsection, as it will help use to gain a deeper understanding of lattice regression.

3.2.2 Interpolation

For interpolation (Garcia and Gupta, 2009) propose a method called *multilinear interpolation*. Any test point $x \in D$ is contained in one of the cells of the lattice, more precisely, any x is surrounded by 2^D vertices from which the lattice output θ will be then interpolated. Fig. 3 shows the simple case of a 2×2 lattice; to evaluate the lattice function, x is

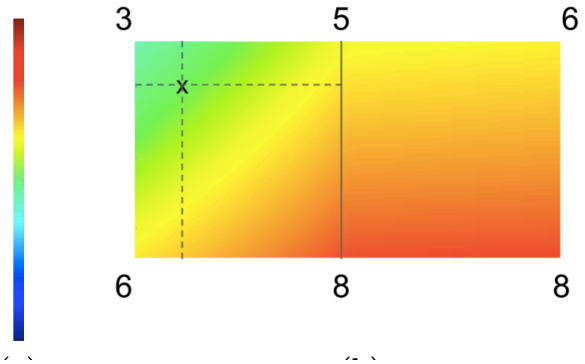


Figure 3: Example for a $D = 2$ lattice function $f(x)$. Assuming (a) would be scaled with $[0, 1]$, then $f(x)$ would be approx. a value of 0.5 (Gupta et al., 2016).

linearly interpolated from parameter values $\theta = (6, 8, 3, 5)$ at the vertices v_1, v_2, v_5, v_6. The

interpolation method therefore consists of a linear combination $\phi_k(x) = [\phi_1(x), \dots \phi_{2^D}(x)]^T$ of the surrounding vertice outputs θ. (Gupta et al., 2016, p. 3) give a very concise graphical description of the functioning of the weighting for low dimensional cases as in Fig. 3:

> The weights on the parameters [at the vertices] are the areas of the four boxes formed by the dotted lines drawn orthogonally through x, with each parameter weighted by the area of the box farthest from it, so that as x moves closer to a parameter the weight on that parameter gets bigger.

To interpolate any point x we first need to be able to index its surrounding lattice cell. To obtain the kth vertex of the lattice cell, that surrounds x, a function $c_k(x) : \mathbb{R}^d \to \mathbb{N}$ (for details on its computation see (Garcia et al., 2012) is defined. Let then $v_{c_k(x)}\ for\ (k = 1, \dots, 2^D)$ be the kth vertex of the lattice cell containing x. The associated weights $\phi_k(x)$ for that cell can then be computed as

$$\phi_k(x) = \prod_{d=0}^{D-1} x[d]^{v_k[d]}(1 - x[d])^{1-v_k[d]}. \tag{4}$$

Note that we assumed a 2^D lattice, in this case $v_k \in \{0, 1\}^D$ can be seen as the k th vertex of the unit hypercube (the logic stays the same for multi-cell lattices, the formulas can be looked up in (Garcia et al., 2012). The exponent functions as a bitwise selector, multiplying one of the two $x[d]$ terms. We can then form the $M \times 1$ sparse weight vector for each observation x_i:

$$\phi(x_i)[d] = \begin{cases} \phi_k(x_i) & \text{if } d = c_k(x_i),\ \text{for } k = 1, \dots, 2^D \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

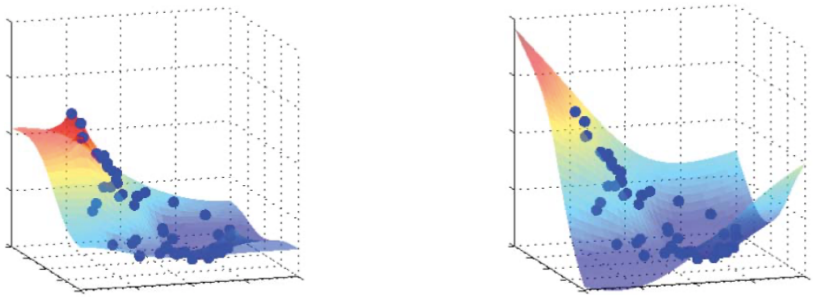
The function value for x_i is then interpolated as $\theta^T\phi(x_i)$; concatenating all these individual vectors yields us a matrix, which can be described as the lattice function $\theta^T\phi(x)$. In addition, we can think of it as a kernel method that maps x to a transformed feature vector $\phi(x)$. Garcia and Gupta chose d-linear interpolation as method, because it can be implemented efficiently and it is the maximum entropy solution to Eq. 1. (Gupta et al., 2016) also accredit the θ parametrization an improved interpretability effect. In addition they propose two new methods for faster interpolation (with *simplex* interpolation requiring $\mathcal{O}(DlogD)$ operations instead of $\mathcal{O}(D2^D)$, as with d-linear interpolation). To complete the lattice regression method we still need to supplement Eq. 2 with a regulizer term.

3.2.3 Regulizer

With the purpose to reduce over-fitting to the training data and to ensure a unique solution to Eq. 2 (Garcia et al., 2012) introduce two kinds of graph regulizers $R(\theta)$. On the one hand there is the *graph Laplacian* regulizer:

$$R_{laplacian}(\theta) = \sum_{adjacent\ v_r,v_s} (\theta_r - \theta_s)^2 = \theta^T \mathbf{K}_L \theta. \tag{6}$$

Here, the sum of squared differences between the lattice values θ at adjacent vertices v_j is penalized. Multiplying Eq. 6 with a regularization paramater $\lambda > 0$ (smoothness/accuracy trade off) and adding it to the regression objective of Eq. 2, this yields a closed form solution $(\phi(x)^T\phi(x)^{-1} + \lambda \mathbf{K}_L)\phi(x)^T y$.



(a) Laplacian **(b)** Hessian

Figure 4: Implementation of two regulizers on the same training data (blue dots). (Garcia et al., 2012)

Garcia and Gupta argue that the Laplacian regulizer is not optimal, as it penalizes the estimation of linear functions. Therefore they propose a regulizer which penalizes the second-order difference in each dimension of the lattice:

$$R_{hessian}(\theta) = \sum_{d=1}^{D} \sum_{\substack{v_r,v_s,v_t \\ adjacent\ in \\ dimension\ d}} ((\theta_t - \theta_r) - (\theta_r - \theta_s))^2 = \theta^T \mathbf{K}_H \theta. \tag{7}$$

They refer to Eq. 7 as *graph Hessian* regulizer. In the same way as Laplacian regulizer it yields a closed form solution to 2:

$$(\phi(x)^T\phi(x)^{-1} + \lambda \tilde{\mathbf{K}}_H)\phi(x)^T y \quad \text{with} \quad \tilde{K}_H = K_H + 10^{-6}\mathbf{I}.$$

What is the difference to the Laplacian? Fig. 4 shows how the Hessian regulizer permits linear extrapolations, whereas the Laplician penalizes them.

Furthermore, (Gupta et al., 2016) propose a third type of regulizer; it functions in a similar way as the Hessian regulizer, making the function more linear. For a concise comparison of all three regulizers, see p. 20.

With the regulizer at hand, the lattice regression objective has the following form:

$$arg \min_{\theta \in \mathbb{R}^M} \sum_{i=1}^{n} \ell(y_i, \theta^T \phi(x_i)) + R(\theta). \tag{8}$$

Regarding the credit score example from Sec. 1 the above objective function is just the base framework. To obtain the full functionality of the TFL calibrated model we will use for the experiment in Sec. 4, Eq. 8 has to be augmented with *monotonicity constraints* and *feature calibration*. We briefly introduce both additions in the following subsection.

3.3 Lattice Framework Extensions

The extension of lattice regression to monotonic functions and jointly learning covariate calibration was proposed by (Gupta et al., 2016). For our case study we will use a TFL linear model which is monotonically constrained and calibrated; therefore we are give a brief overview on both in the next two subsections.

3.3.1 Monotonicity

For the problem setting of lattice regression Gupta et al. give the following definition for monotonicity:

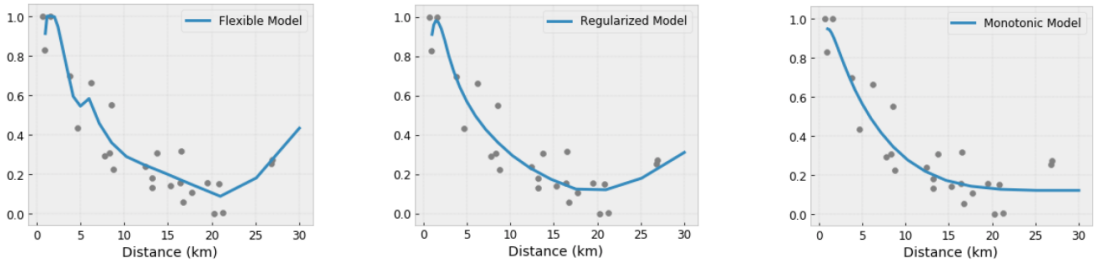
> **Definition.** *A function $f(x)$ is monotonically increasing with respect to feature d if $f(x_i) \geq f(x_j)$ for any two feature vectors $x_i, x_j \in \mathbb{R}^D$ where $x_i[d] \geq x_j[d]$ and $x_i[m] = x_j[m]$ for $m \neq d$.*

Monotonic constraints are applied to machine learning problems to impose prior knowledge and also regularize the estimated functions. Fig. 5 shows an example of an applied constraint. Gupta et al. give an extinsive overview of the related literature concerning the learning of monotonic functions.

For lattice regression, the goal is to ensure that a monotonic function is learned. Therefore some constraints have to be imposed on lattice output parameters θ during the learning process. For a 2^D lattice to be monotonically increasing in the dth covariate Gupta et al. give the following lemma for the monotonicity constraints (p. 12):

Lemma. *Let $f(x) = \theta^T \phi(x)$ for $x \in [0,1]^D$ and $\phi(x)$ given in Eq. 4. The partial derivative $\partial f(x)/\partial x[d] > 0$ for fixed d and any x iff $\theta k' > \theta k$ for all k, k' such that $v_k[d] = 0, v'_k[d] = 1$ and $v_k[m] = v'_k[m]$ for all $m \neq d$.*

Monotonicity is established by imposing pairwise linear inequality constraints on the lattice output θ: for each pair of adjacent lattice parameters θ_r and θ_s the inequality $\theta_s > \theta_r$ has to hold. As a result the estimated function increases in a certain direction, if the lattice parameters increase in that direction.



(a) **(b)** **(c)**

Figure 5: The monotonic model (c) ensures that the output only decreases with respect to an input; models (a) and (b) fail in this regard (`https://www.tensorflow.org/lattice/overview`).

To implement the constraints in the regression objective, Gupta et al. then relax strict monotonicity to monotonicity 'by allowing equality in the adjacent parameter constraints' (p. 13). The updated monotonic lattice regression objective then looks the following:

$$arg\ \min_{\theta} \sum_{i=1}^{n} \ell(y_i, \theta^T \phi(x_i)) + R(\theta) \text{ s.t. } A\theta \leq b. \tag{9}$$

The pairwise constraints for the lattice output θ is written as $A\theta \leq b$, whereby A is a sparse matrix with one row per constraint and one 1 and -1 per row. A enables the specification of each individual covariate, wheter it should be constrained or unconstrained. To further constrain the fitted function in linear ways (for instance by $f(x) \geq 0$) additional linear constraints can be imposed through $A\theta \leq b$.

3.3.2 Calibration

As a way to increase accuracy, (Gupta et al., 2016) propose a method for transforming each covariate before they are supplied to the lattice function $f(x)$. This poses an alternative to refining the lattice by increasing the hyperparameter M_d, as this would be inefficient on a

larger scale. *Feature calibration* is an automated one-dimensional pre-processing operation $c_d[x_d]$, calibrating each of the d covariates of $x \in D$. For continous data the authors present a one-dimensional monotonic piecewise linear function. For categorical data they propose a function for mapping each category to a real value in $[0, M_d - 1]$. Furthermore there is a suggestion of two methods to handle missing values by using a calibration approach. Now, to learn a calibrated monotonic lattice, calibration functions $c_d(\cdot)$ and lattice parameters θ are simultaneously optimized. This requires the following adjustment of the regression objective:

$$arg \min_{\theta,\alpha} \sum_{i=1}^{n} \ell(y_i, \theta^T \phi(c(x_i, \alpha))) + R(\theta) \text{ s.t. } A\theta \leq b \text{ and } \tilde{A}\alpha \leq b. \tag{10}$$

The vector function $c(x; \alpha)$ maps a feature vector x to the lattice function $\theta^T \phi(x)$ and entails the dth component function $c_d(x[d]; \alpha^{(d)})$, which is a calibration function depending on the parameter $\alpha^{(d)}$ and the dth component of x. As before A denotes the monotonicity constraint, while $\tilde{A}$ does the same for each pair of adjacent calibration parameters of the calibration functions $c(x; \alpha)$. We leave our review of lattice calibration at that. For a far more detailed description see Gupta et al. (2016); furthermore an neural network implementation (*deep lattice networks*) of the theory briefly sketched in this section can be found in (You et al., 2017).

4 Experiment Set-up

In this section we give an overview on the data used for the experiment, its processing and how it was split. More importantly, we outline the specifications of the modeling process and the evaluation method for performance assessment.

4.1 Data Sets and Preprocessing

To conduct our experiment we use four credit scoring data sets. There are on the one hand two data sets obtained from *Kaggle*: the 2010 *PAKDD data mining challenge* data set[2] (PAK) and the *Give Me Some Credit competition* data set[3] (GMC). On the other hand we make use of two well-known UCI Machine Learning Library data sets, the *Default of Credit Card Clients* data set (TCD) from Taiwanese bank customers (Yeh and Lien, 2009) and the *South German credit* data set (GER), which is a reviewed version of the very common *German credit* data set (Groemping, 2019). A brief overview on the data is given in Tab. 2. Each data set contains customer data from financial institutions. All sets are composed of a mixture of socio-economic information from the customer (i.e. age, education, maritial status), information from their loan inquiry (i.e. amount, duration) and an assessment of whether the customer failed to repay his loan.

Data Set	Prior default risk	Observations	Number of variables	Number of selected vars.	Missing Values
GER	0,30	1,000	21	16	No
GMC	0,07	150,000	12	11	Yes
PAK	0,26	50,000	54	11	Yes
TCD	0,22	30,000	25	17	No

Table 2: Overview data sets

The first step for data processing is to make a decision about variable selection, as our lattice models requires a low amount ($n \leq 20$) of covariates.[4] The main criteria for this selection are (*i.*) correlation, see the exemplary correlation plot in Appendix Fig. 8, (*ii.*) and comparability, which aims to have similar variables for all data sets. Tab. 2 shows the

[2] `https://www.kaggle.com/c/pakdd2010-dataset/`

[3] `http://www.kaggle.com/c/GiveMeSomeCredit`

[4] In addition, the computational resources for performing the experiment were rather restriced.

reduction of variables for each data set. As a next step we apply standard preprocessing operations as imputation of missing values by mean/mode replacement. In addition we transform the data type of every variable to be numeric, since the lattice model requires tensor shaped input data. We do not normalize the data, as none of the models requires such an operation. For the partitioning of the data we apply a simple 80/20 train/test split, no validation or cross validation, to keep computing effort low.

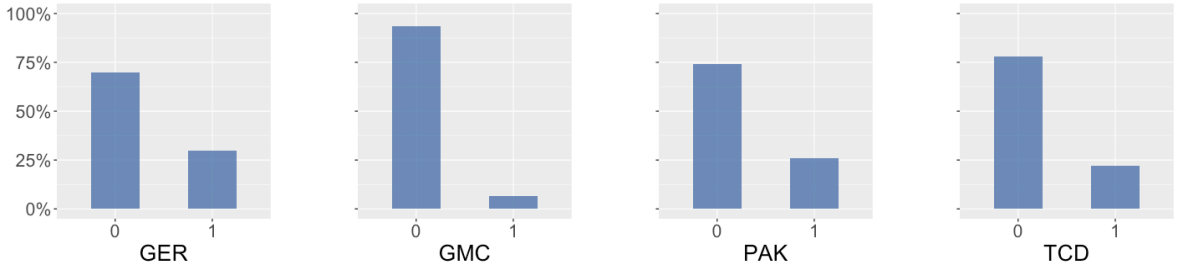


Figure 6: The distribution of credit default for each data set.

4.2 Modeling and Evaluation

The goal of this thesis is to explore lattice based models. To asses its performance, we set up an experiment in which we compare a nonlinear generalized additive model (GAM), also referred to as lattice model, and a random forrest classifier. The algorithms are applied in a credit scoring domain on a classification problem: each of the above introduced data sets contains a binary class label for each client that determines if there is a prior risk of some kind of default or not (see Fig. 6). We renamed the target variable for each data set to 'default' and since we are facing a binary outcome we set the value for 'customer likely defaults' to $y = 1$ and for 'customer likely not defaults' to $y = 0$. We chose a random forrest classifier for the comparison, as it is commonly used for credit scoring (Baesens et al., 2003), (Lessmann et al., 2015). Due to computational restrictions we forwent tuning the hyperparameters of the models. To deploy the lattice model we set up an interface to call the Tensor Flow Lattice Python library, as the experiment was run in an R environment. A full description of each model can be found in the accompanying R-script. The following gives a brief overview on the most important parameters and aspects of each model:

Calibrated Lattice Model

Among a set of hyperparameters our, 'nonlinear generalized additive model (GAM), also called a calibrated linear model in the TensorFlow Lattice' (Wang and Gupta, 2020, Sec. 8),

requires a detailed specification for calibration. As we saw in Sec. 3.3.2, a lattice model first applies piecewise-linear and categorical calibration on each input variable. The calibration function $c_d(x[d]; \alpha^{(d)})$ is parametrized by K ($\alpha^{(d)} \in [0, M_d - 1]^K$, with K differing for continous and categorical variables), which in simple terms determines the spacing of $c_d(\cdot)$ (Gupta et al. (2016) give a detailed explanation of the calibrator on pp. 26). The amount of keypoints K set and the type of spacing (by quantiles, linearly or manually) has to be specified for every variable of each data set. Our decision making about the parametrization of $c(\cdot)$ was guided by the distribution of the individual variables; if e.g. a variable had a strong skew and outliers, we would pick a higher value for K and a quantile spacing. The specification for every variable and data set can be looked at in the feature configs Python file.

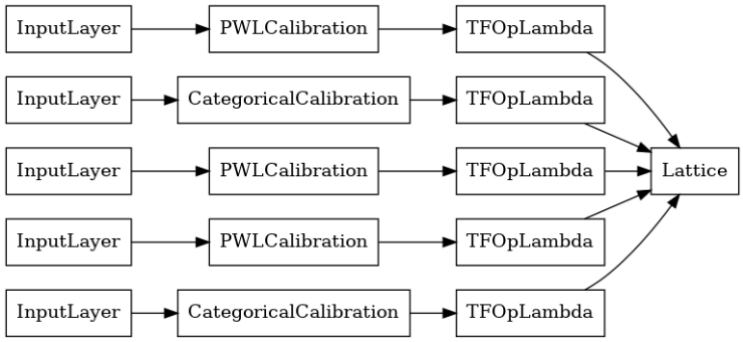


Figure 7: A simplified version of our model

There are two further important per-variable configurations entailed in this file: lattice size and monotonicity. The first we kept mostly at a default value of 2 for the vast majority of variables, mainly to keep computational effort low. For monotonicity we again looked at variable distribution behaviour, to decide if it makes sense to apply monotonicity and whether it should be in- or decreasing. Simple charts of the variables as in Appendix Fig. 9 were helpful for decisions.

After calibration a lattice layer non-linearly fuses the calibrated variables, see Fig. 7. As a regulizer for the calibration we chose the Hessian, and for making the lattice more linear we used the Torsion regulizer, both set to a value of 0.01. Further parameters included *binary crossentropy* loss function and *Adam* optimizer; a *learning rate* of 0.01, a rather large *batch size* of 128 and a rather small *epoch size* of 25, both for computational reasons.

Random Forrest

Our random forest model of choice contains three potential tuning parameters: a value for the number of trees, a value for the number of data points that are required for a node to

split and a value for the number of predictors that will be randomly sampled at each split. We set the tree parameter at 1000, the minimum node size parameter to 3 and the last one at *null*. A more detailed explanation of the model can be found in (Breiman, 2001).

Evaluation

Both of the introduced models were then applied to all of the credit scoring data sets. For the assessment of each classifier we chose AUC as evaluation metric, since it is a well established metric for this kind of classification problem (Lessmann et al., 2015).

For the comparison of the performance of two classifiers on multiple datasets it is recommended to perform a statistical test on the obtained evaluation metrics, rather then just averaging over them (Japkowicz and Shah, 2011). For our comparison of the AUC values of both models we therefore conducted *Wilcoxon's Signed-Rank test*, following the approach of (Japkowicz and Shah, 2011, Sec. 6.6). For this test the H_0-hypothesis is that both classifiers perform equally as good; we are thus going to try to show, that one classifier performs better than the other one.

Wilcoxon's Signed-Rank test works the following way: for each pair of AUC values the difference d_i is taken. These get ranked, respectively their absolute values; if a tie occurs, an average rank will be assigned to the tied values. Then a sum of ranks is calculated:

$$W_{s1} = \sum_{i=1}^{n} I(d_i > 0)\text{rank}(d_i) + \frac{1}{2}\sum_{i=1}^{n} I(d_i = 0)\text{rank}(d_i),$$
$$W_{s2} = \sum_{i=1}^{n} I(d_i < 0)\text{rank}(d_i) + \frac{1}{2}\sum_{i=1}^{n} I(d_i = 0)\text{rank}(d_i).$$

With these sums a test statistic can be calculated as $T_{wilcox} = \min(W_{s1}, W_{s2})$. To verify the rejection of H_0, critical values for $n < 25$ can be looked up from a table, otherwise the T_{wilcox} distribution can be approximated. For a more detailed description of the test procedure see (Japkowicz and Shah, 2011, p. 234).

5 Results

As summary of the results of our experiment is provided in Tab. 3. On a first glance there is no larger difference in the performance of both classifier, with random forrest doing slightly better in most of cases. But the averaging over the AUC_{ts} values is deceiving. Applying Wilcoxon's Signed-Rank test on the result leaves us with a p-value of 0.25; we therefore cannot reject the H_0-hypothesis. Based on the given data and employing the above sketched models we cannot state that one classifier performs better then the other. This result aligns with those of (Gupta et al., 2016, Sec. 10) and (Wang and Gupta, 2020, Sec. 8), who both test calibrated lattice models against a random forrest classifier. The former tune their hyperparameters, our experiment set up is more comparable to the one of Wang & Gupta. As we saw in Sec. 3.1, they also used the TCD data set. Our experiment can therefore partially be seen as a confirmation of their experiment, since we only reproduced it with a different classifier for comparison.

	Lattice		**Tree**		Diff
	AUC_{tr}	AUC_{ts}	AUC_{tr}	AUC_{ts}	AUC_{ts}
GER	0.7523	**0.7737**	1.0000	0.7643	0.0094
GMC	0.8367	0.8438	0.9998	**0.8585**	-0.0146
PAK	0.5481	0.5422	0.9371	**0.5733**	-0.0311
TCD	0.7316	0.7271	0.9997	**0.7728**	-0.0458
Average	0.7171	0.7216	0.9841	0.7422	

Table 3: Resulting *AUC* values from the experiment

For improving the experiment set-up to hopefully obtain even more sound results, a couple of measures could be taken:

- The statistical test could be altered, so that we actually reject the H_0-Hypothesis. An alternative would be an Equivalence test (Walker and Nowacki, 2011).
- The number of credit scoring data sets could be increased.
- The data could be split in a more sophisticated manner, for instance by cross-validation.
- Other types of classifiers could be introduced for comparison
- The grid could be tuned for all of the hyperparameters.

6 Conclusions

In this paper we gave a brief introduction into the lattice regression framework and its extensions. We furthermore gave an example of how covariate shift can affect credit scoring applications and proposed to use a lattice model as a mitigation strategy. In our case study we then showed that a monotonic calibrated lattice model does not perform worse on credit data then a default random forrest model.

The proposed ameliorations in Sec. 5 can be seen as a first step to improve the study. As a next step it would be interesting to compare the model with one or more covariate shift correction algorithms presented in Sec. 2. In addition, another feature of the lattice model should be explored further: the proclaimed enhanced interpretability of the model through the lattice structure (Gupta et al., 2016). Recently, interpretability is becoming a topic of concern, as so called 'black box'-models are criticized for being unsuitable to be used in ethically crucial situations due to lack of transparency (Rudin, 2019).

Throughout the paper it should have shown that covariate shift touches deeply on the problem of inductive reasoning. When (Shimodaira, 2000) introduced the concept of covariate shift, he claimed that is it exclusively caused by model misspecification and that it would disappear for a correct model. But when do we ever know the correct specification of a model? 'In reality, however, good fitting models with a modest number of parameters are hard to obtain except for very simple applications.' (Quionero-Candela et al., 2009, p. 202) says Shimodaira. The question 'How do we justify generalization for our model from train data to test data?' will continously loom in the background.

References

Baesens, B., T. Van Gestel, S. Viaene, M. Stepanova, J. Suykens, and J. Vanthienen (2003): "Benchmarking state-of-the-art classification algorithms for credit scoring," *Journal of the operational research society*, 54, 627–635.

Ben-David, S. (2009): "On the Training/Test Distributions Gap: A Data Representation Learning Framework," *Dataset shift in machine learning*, 73–84.

Ben-David, S., J. Blitzer, K. Crammer, F. Pereira, et al. (2007): "Analysis of representations for domain adaptation," *Advances in neural information processing systems*, 19, 137.

Ben-David, S. and R. Schuller (2003): "Exploiting task relatedness for multiple task learning," in *Learning theory and kernel machines*, Springer, 567–580.

Bickel, S., M. Brückner, and T. Scheffer (2009): "Discriminative learning under covariate shift." *Journal of Machine Learning Research*, 10.

Breiman, L. (2001): "Random forests," *Machine learning*, 45, 5–32.

Garcia, E., R. Arora, and M. R. Gupta (2012): "Optimized regression for efficient function evaluation," *IEEE Transactions on Image Processing*, 21, 4128–4140.

Garcia, E. and M. Gupta (2009): "Lattice regression," *Advances in Neural Information Processing Systems*, 22, 594–602.

Globerson, A., C.-H. Teo, A. Smola, S. Roweis, et al. (2009): "An adversarial view of covariate shift and a minimax approach," in *Dataset shift in machine learning*, MIT Press.

Gretton, A., A. Smola, J. Huang, M. Schmittfull, K. Borgwardt, and B. Schölkopf (2009): "Covariate shift by kernel mean matching," *Dataset shift in machine learning*, 3, 5.

Groemping, U. (2019): "South German credit data: Correcting a widely used data set," *Rep. Math., Phys. Chem., Berlin, Germany, Tech. Rep*, 4, 2019.

Gupta, M., A. Cotter, J. Pfeifer, K. Voevodski, K. Canini, A. Mangylov, W. Moczydlowski, and A. Van Esbroeck (2016): "Monotonic calibrated interpolated look-up tables," *The Journal of Machine Learning Research*, 17, 3790–3836.

Huang, J., A. Gretton, K. Borgwardt, B. Schölkopf, and A. Smola (2006): "Correcting sample selection bias by unlabeled data," *Advances in neural information processing systems*, 19, 601–608.

Japkowicz, N. and M. Shah (2011): *Evaluating learning algorithms: a classification perspective*, Cambridge University Press.

Johansson, F. D., N. Kallus, U. Shalit, and D. Sontag (2018): "Learning weighted representations for generalization across designs," *arXiv preprint arXiv:1802.08598*.

Kanamori, T., S. Hido, and M. Sugiyama (2009a): "A least-squares approach to direct importance estimation," *The Journal of Machine Learning Research*, 10, 1391–1445.

Kanamori, T., T. Suzuki, and M. Sugiyama (2009b): "Condition number analysis of kernel-based density ratio estimation," *arXiv preprint arXiv:0912.2800*.

Kato, M., M. Uehara, and S. Yasui (2020): "Off-policy evaluation and learning for external validity under a covariate shift," *arXiv preprint arXiv:2002.11642*.

Kull, M. and P. Flach (2014): "Patterns of dataset shift," in *First International Workshop on Learning over Multiple Contexts (LMCE) at ECML-PKDD*.

Lessmann, S., B. Baesens, H.-V. Seow, and L. C. Thomas (2015): "Benchmarking state-of-the-art classification algorithms for credit scoring: An update of research," *European Journal of Operational Research*, 247, 124–136.

Miao, Y.-Q., A. K. Farahat, and M. S. Kamel (2015): "Ensemble kernel mean matching," in *2015 IEEE International Conference on Data Mining*, IEEE, 330–338.

Moreno-Torres, J. G., T. Raeder, R. Alaiz-RodríGuez, N. V. Chawla, and F. Herrera (2012): "A unifying view on dataset shift in classification," *Pattern recognition*, 45, 521–530.

Nair, N. G., P. Satpathy, J. Christopher, et al. (2019): "Covariate Shift: A Review and Analysis on Classifiers," in *2019 Global Conference for Advancement in Technology (GCAT)*, IEEE, 1–6.

Quionero-Candela, J., M. Sugiyama, A. Schwaighofer, and N. D. Lawrence (2009): *Dataset shift in machine learning*, The MIT Press.

Rudin, C. (2019): "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead," *Nature Machine Intelligence*, 1, 206–215.

Shimodaira, H. (2000): "Improving predictive inference under covariate shift by weighting the log-likelihood function," *Journal of statistical planning and inference*, 90, 227–244.

Sterkenburg, T. F. and P. D. Grünwald (2020): "The no-free-lunch theorems of supervised learning," .

Storkey, A. (2009): "When training and test sets are different: characterizing learning transfer," *Dataset shift in machine learning*, 3–28.

Sugiyama, M. (2010): "Superfast-trainable multi-class probabilistic classifier by least-squares posterior fitting," *IEICE Transactions on Information and Systems*, 93, 2690–2701.

Sugiyama, M. and M. Kawanabe (2012): *Machine learning in non-stationary environments: Introduction to covariate shift adaptation*, MIT press.

Sugiyama, M., M. Krauledat, and K.-R. Müller (2007): "Covariate shift adaptation by importance weighted cross validation." *Journal of Machine Learning Research*, 8.

Sugiyama, M., T. Suzuki, S. Nakajima, H. Kashima, P. von Bünau, and M. Kawanabe (2008): "Direct importance estimation for covariate shift adaptation," *Annals of the Institute of Statistical Mathematics*, 60, 699–746.

Walker, E. and A. S. Nowacki (2011): "Understanding equivalence and noninferiority testing," *Journal of general internal medicine*, 26, 192–196.

Wang, S. and M. Gupta (2020): "Deontological Ethics By Monotonicity Shape Constraints," .

Yamada, M., M. Sugiyama, G. Wichern, and J. Simm (2011): "Improving the accuracy of least-squares probabilistic classifiers," *IEICE transactions on information and systems*, 94, 1337–1340.

Yamazaki, K., M. Kawanabe, S. Watanabe, M. Sugiyama, and K.-R. Müller (2007): "Asymptotic bayesian generalization error when training and test distributions are different," in *Proceedings of the 24th international conference on Machine learning*, 1079–1086.

Yeh, I.-C. and C.-h. Lien (2009): "The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients," *Expert Systems with Applications*, 36, 2473–2480.

You, S., D. Ding, K. Canini, J. Pfeifer, and M. Gupta (2017): "Deep lattice networks and partial monotonic functions," *arXiv preprint arXiv:1709.06680*.

A Figures

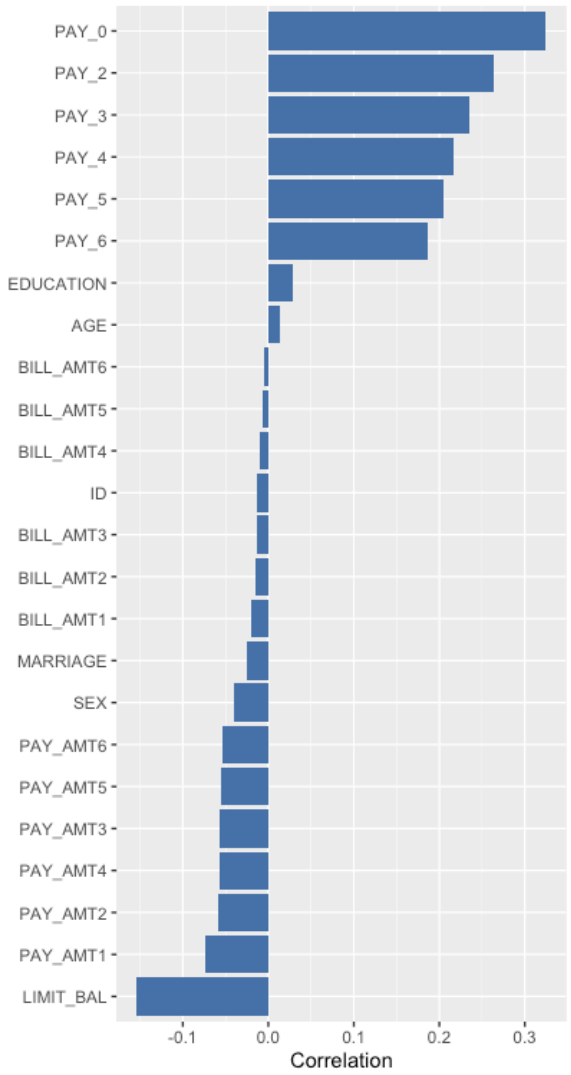


Figure 8: The Figure shows the correlation of the dependant with the independant covariates for TCD. We removed all 'BILL' variables, 'ID' and 'MARRIAGE', but kept 'AGE' for comparability.

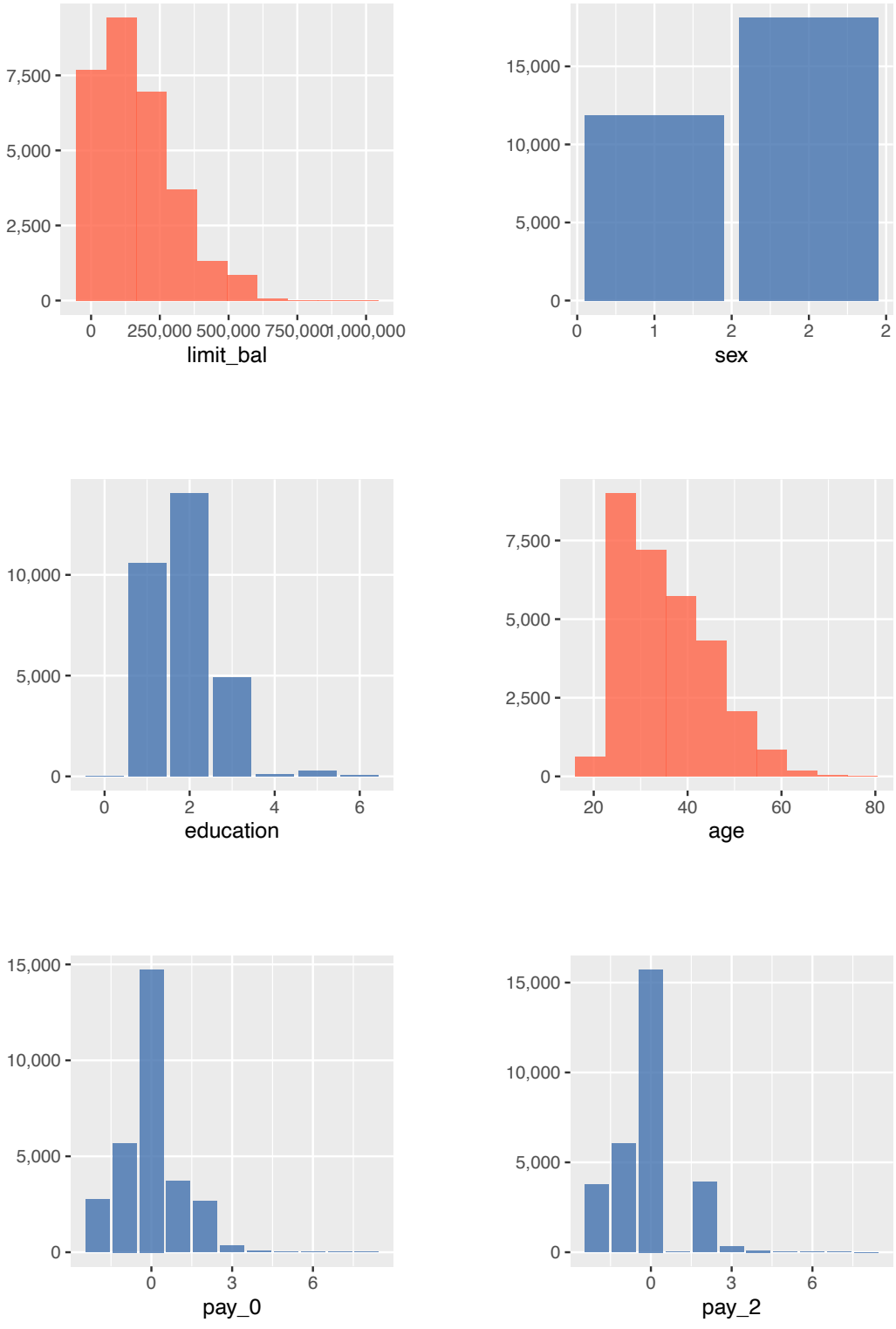


Figure 9: Example plots for TCD data set

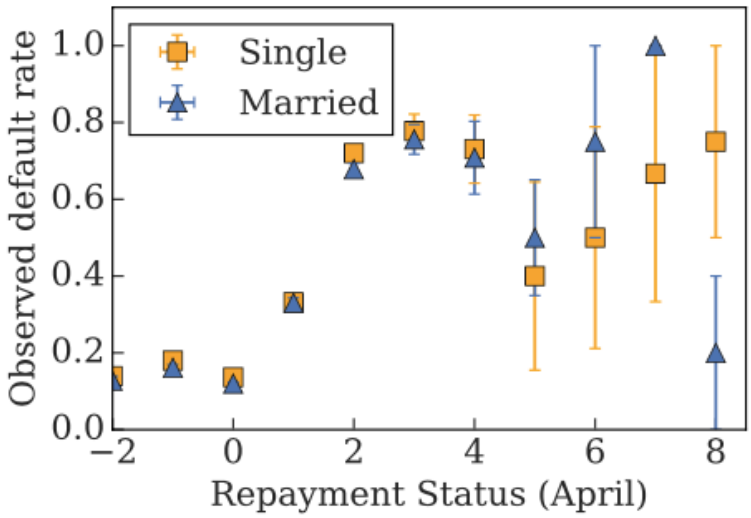


Figure 10: Training examples for credit data (Wang and Gupta, 2020)