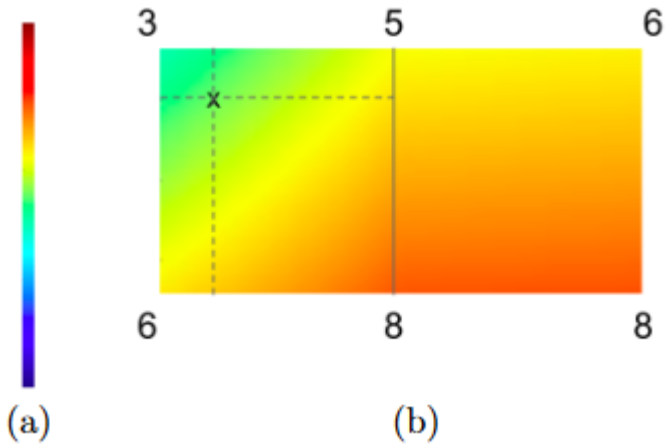
Monotonic Calibrated Interpolated Look-Up Tables

Author: MAya Gupta Tags: Primary Type: Paper URL: http://jmlr.org/papers/v17/15-243.html

1. Introduction

key interpretability issue in practical ML: can the learnd model be guaranteed to be monotonic wrt some input features → this paper proposes learning monotonic, efficient and flexible functinons by constraining and calibrating interpolated look-up tables in a structural risk minimization framework - the parameters of the ILT (interpolated look-up table) are the values of the function, regularly spaced in the input space, and these values are interpolated to compute $f(x)$ for any x.



ILT are a classic strategy for representing low-dim functions (see most backs of textbooks). Here, ILT are defined over much larger feature spaces. Using efficient linear interpolation (*simplex interpolation*) the interpolation fo a D-dimensional LuT can be computed in $\mathcal O (D \log(D))$ time.

Estimating the parameters of a ILT usisng structural risk minimizaion was proposed by Garcia and Gupta (2009) and called *lattice regression*.

Lattice regression: can be viewed as a kernel method that uses the explicit nonlinear feature transformation formed by mappin an input $x \in [0,1]^D$ to a vector of linear interpolation weights $\Phi(x) \in \Delta^{2^D}$ over the 2^D vertices of the LUT-cell that contains x, where Δ denotes the standard simplex. The function is the linear in these transformed featurues: $f(x) = \Theta^T \Phi(x)$. The LUT parameters are referred to as the *lattice*, and the interpolated LuT $f(x)$ as *lattice function*.

2. Related Work

2.1. Related Work in Interpretable Machine Learning

Tow key themes of the prior work on interpretable machine learning are (i) interpretable function classes, and (ii) preferring simpler functions within a function class.

2.1.1. Interpretable Function Classes

relatively interpretable are *decision trees and rules*, *naive Bayes classifiers, linear models, gerneralized additive models* and some *kernel methods* as well as *interpolated look-up tables*

2.1.2. Prefer Simpler Functions

Chose simpler functions within a function class, optimizin an objective of the form: *minimize empirical error and maximize simplicity,* where simplicity is defined as some manifestation of Occam's Razor or variant of Kolmogorov complexity. Inlcudes also model selection criteria like *AIC* or *BIC*, sparsity regularizers as *sparse linear regression models*, and feature selection methods. Sparsity-based approcheeas can provide regularization, but may create a tradeoff between itnerpretability and accuracy.

2.2. Related Work in Monotonic Functions

A function $f(x)$ is monotonically increasing with respect to feature d if $f(x_i) \ge f(x_j)$ for any two feature vectors $x_i, x_j \in \R^D$ where $x_i[d] \ge x_j[d]$ and $x_i[m] = x_j[m]$ for $m \ne d$.

Here the related work is organized by the type of machine learning, but it could also be organized by strategy, which mostly falls into one of four categories:

- Constrain a more flexible function class to be monotonic, such as linear functions with positive coefficienets
- Post-process by pruning or reduction monotonicity violations after training
- Penalize monotonocity violations by pairs of samples or sample derivatives when training
- Re-label samples to be monotonic before training

2.2.1. Monotonic Linear and Polynomial Functions

Lin. functions can easily be constrained to be monotonic in certain inputs by requiring thee corresponding slope coefficients to be non-negative. For polyn. functions holds the same. This is only a sufficient and not a necessary condition. The problem of checking whether a particular choice of polyn. coefficients produces a monotonic function requires checking whether the polynomials's derivative (also a polynomial) is positive everywhere == checking if the derivative has any real roots (comp. challenging)

2.2.2 Monotonic Splines

Here we extend lattice regression (a spline method with fixed knots on a regular grid and a linear kernel) to be monotonic.

2.2.3. Monotonic Decision Trees and Forests

Stumps and forests of stumps are easily constrained to be monotonic. For deeper or broader trees, all pairs of leaves must bee checked to verify monotonicity. Non-monotonic trees can be pruned to be monotonic.

2.2.4. Monotonic Support Vector Machines

With linear kernel it's easy. With nonlinear kernel it's challenging. Lauer and Bloch (2008) did it by constraining the derivative of the fucntion at the training samples, Riihimäki and Vehtari (2010) used the same strategy to encourage more monotonic gaussian processes.

2.2.5. Monotonic Neural Networks

Here many different methods are given in the literature.

2.2.6. Isotonic Regression and Monotonic Nearest Neighbour

Isotonic regression re-labels the input samples with values that are monotonic and close to the original labels (old approach Barlow (1972)). Also other methods are tried out, see Literature.

3. Review of Lattice Regression

see Garcia and Gupta (2009) or (Garcia et al. 2012)

4. Monotonic Lattices

constraining lattice regression to learn monotonic functions

4.1. Monotonicity Constraints for a Lattice

If the lattice values increase in a given direction, then the function increases in that direction. Therefore, one must check that $\Theta_s \ge \Theta_r$ for each pair of adjacend LuT parameters Θ_r and Θ_s. If all features are specified to be monotonic for a 2^D lattice, this results in $D 2^{D-1}$ pairwise linear inequality constraints to check.

These same pairwise linear inequality constraints can be imposed when learing the parameters Θ to ensure a monotonic function is learned. The following lemma establueshes that these constraints are sufficient and necessary for a 2^D lattice to be monotonically increasing in the d-th feature:

- **Lemma 1 (Monotonoicity Constraints):** *Let $f(x) = \theta^T \Phi(x)$ for $x \in [0,1]^D$ and $\Phi(x) = \prod_{d=0}^{D-1} x[d]^{v_k[d]} (1 - x[d])^{1 - v_k[d]}$. The partial derivative $\frac{ \partial f(x) } {\partial x[d]} \ge 0$ for fixed d and any x iff. $\theta_{k'} > \theta_k$ for all k, k' sucht that $v_k[d] = 0, v_{k'} = 1$ and $v_l[m] = v_{k'}[m]$ for all $m \ne d$.*
- **Proof:** Given in the paper!

4.2. Monotonic Lattice Regression Objective

The set of pairwise constraints can be expressed as $A \theta \se b$ (if constraints are relaxed to include equality constraints) for the appropriate sparse matrix A with one 1 and one -1 per row of A, and one row per constraint.

The monotonic lattice regression objective is convec with linear inequality constraints:

$$\arg \min_{\theta} \sum_{i=1}^n \mathcal l(y_i, \theta^T \phi(x_i)) + R(\theta), \quad s.t. A\theta \le b.$$

$R(\theta)$ is a regularizer on the lattice parameters. Additional linear constraints can be included in $A\theta \le b$ to also constrain the fitted function in other practical ways, such as $f(x) \in [0, 1]$ or $f(x) \ge 0$.

5. Faster Linear Interpolation

Computing linear interpolation weights with the classical approach requires $\mathcal O(D2D^D)$ operations. Multilinear interpolation can compute the weights in $\mathcal o(2^D)$ operations. The *simplex interpolation* is even faster with $\mathcal O(D \log(D))$ operations.

- **Fast Multilinear Interpolation:** Makes use of the fact, that much of the computation can be shared between the different weights.

- **Simplex Linear Interpolation:** was proposed be Kasson et al. (1993), Idea: find the simplices that sourround the test point x and compute the linear interpolation only with these simplices. The problem here is that simplex interpolation is rotation dependent, which can cause problems

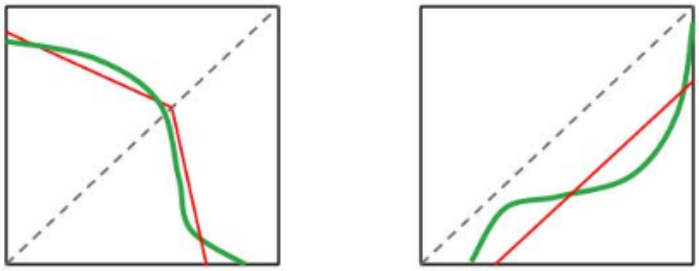


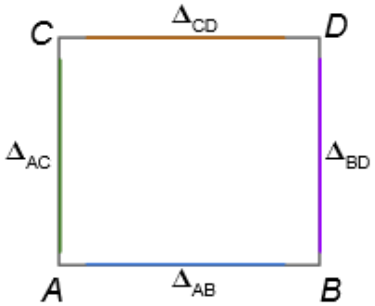
Figure 5: Illustrates rotational dependence of simplex interpolation for a 2×2 lattice and its impact on a binary classification problem. Green thick line denotes the true decision boundary of a binary classification problem. Red thin lines denote the piecewise linear decision boundary fit by lattice regression using simplex interpolation. Dotted gray line separates the two simplices; the function is locally linear over each simplex. **Left:** The true decision boundary (green) crosses the two simplices. The simplex decision boundary (red) has two linear pieces and can fit the green boundary well. **Right:** The same green boundary has been rotated ninety degrees, and now lies entirely in one simplex. The simplex decision boundary (in red) is linear within each simplex, and hence has less flexibility to fit the true green decision boundary.

6. Regularizing the Lattice Regression to be more Linear

Here, a new regularizer (additionally to the Graph Laplacian and Graph Hessian) is introduced. It encourages the fitted function to be more linear by penalizing differences in parallel edges → *torison regularizer*.

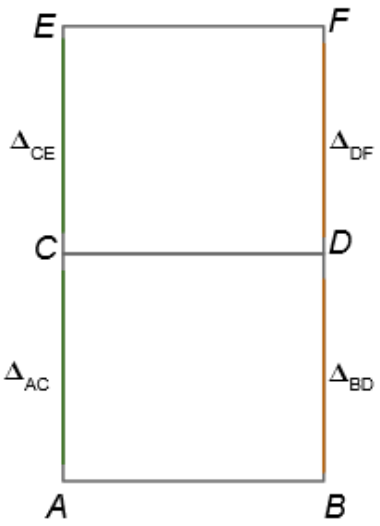
Graph Laplacian:
flatter function

Penalizes:
$(A\text{-}C)^2 + (A\text{-}B)^2 + (C\text{-}D)^2 + (B\text{-}D)^2$
$= \Delta_{AC}^2 + \Delta_{AB}^2 + \Delta_{CD}^2 + \Delta_{BD}^2$



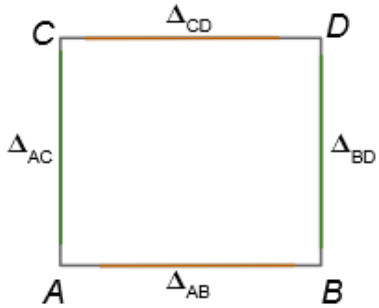
Graph Hessian:
more linear function

Penalizes:
$((A\text{-}C) - (C\text{-}E))^2 + ((B\text{-}D) - (D\text{-}F))^2$
$= (\Delta_{AC} - \Delta_{CE})^2 + (\Delta_{BD} - \Delta_{DF})^2$



Torsion Regularizer:
more linear function

Penalizes:
$((A\text{-}C) - (B\text{-}D))^2 + ((A\text{-}B) - (C\text{-}D))^2$
$= (\Delta_{AC} - \Delta_{BD})^2 + (\Delta_{AB} - \Delta_{CD})^2$



7. Jointly Learning Feature Calibration

Arbitrary bounded functions can be learned with a sufficient fine-grained lattice, but the total number of parameters grows fast. In practive if the features are first each transformed appropriatly, then many problems require only a 2^D lattice to capture the feature interaction. Instead of relying on a user to determine how to best transform each feature, we automate this feature pre-processsing by augmenting the function class with D one-dimensional transformations $c_d(x[d])$ that we jointly learn with the lattice.

7.1. Calibrating Continous Features

Similar to the work of Howard and Jebara (2007)-

The joint estimation makes the objective non-convex. To simplify estimating the parameters, we treat the number of changepoints C_d for the d-th feature as a hyperparameter, and fix the C_d changepoint locations (also called *knots*) at equally-spaced quantiles of the feature values. The changepoint values are then optimized jointly with the lattice parameters.

7.2. Calibrating Categorical Features

If the d-th feature is categorical, use a calibration function $c_d(.)$ to map each category to a real value in $[0, M_d - 1]$.

8. Calibrating Missing Data and Using Missing Data Vertices

Two supervised approches:

- *supervised imputation:* calibrate a missing data value for each missing feature
- *missing data vertices:* give missing date its own vertices in the lattice

9. Large-Scale Training

For convec loss functions $\mathcal l(\theta)$ and convex regularizers $R(\theta)$, any solver for convex problems with linear inequality constraints can be used to optimize the lattice parameters θ. For large n and even for relatively small D, training the calibrated monotonic lattice is challenging due to the number of inequality constraints, the number of terms in the graph reegularizers and the non-convexity created by using calibration functions. ⇒ use of *stochastic gradient descent* methods (SGD)

9.1. SGD and Reducing Variance of the Subgradients

One finds the corresponding subgradient of $\arg \min_{\theta} \sum_{i=1}^n \mathcal l(y_i, \theta^T \phi(x_i)) + R(\theta), \quad s.t. A\theta \le b.$ for the training sample (x_i, y_i), and takes a tiny step in its negative gradient direction. A straightforward SGD implementation would use the subgradient:

$$\Delta = \nabla_{\theta} \mathcal l(\theta^T \Phi(x_i), y_i) + \nabla_{\theta} R(\theta)$$

Ideally, thesee subgradients are cheap to compute. The computational cost ist dominated by computing the regularizers (if any graph regularizer is used).

This subgradient is a realization of a stochastic subgradient whose expectation si equal to the true gradient. The number of iterations needed for SGD convergence depends on the squared Euclidean norm of the stochastic subgradients. The expeected squared norm can be decomposed into two parts:

- the *squared expected subgradient magnitude:* here, little can be done
- the *variance:* can improve the tradeoff between comp. cost of each subgradient and the variance of the stochastic subgradients.

Two strategies are described next.

9.1.1. Mini-Batching

Take the mean gradient of all gradients inside the batch $\mathcal S_l$ → reduces the variance and increases the computational cost. For sufficiently small $\vert \mathcal S_l \vert$, this is a net win because differentiating the regularizer is the dominant computational term.

9.1.2. Stochastic Subgradients for Regularizers

Reduce the comp. cost by randomly sampling the additive term of the regularizer, for regularizers that can be expressed as a summ of terms: $R(\theta) = \sum_{j=1}^m r_j(\theta)$. This makes the subgradient's regularizer term stochastic and hence increases the subgradients variance, but it decreases the comp. effort.

9.2 Parallelizing and Averaging

For a large number of training samples n, one can split the n training samples into K sets, then independently and in-parallel train a lattice on each of the K sets. Once trained, the vector lattice parameters for the K lattices can simply be averaged.

9.3. Jointly Optimizingg Lattice and Calibration Functions

Let $c_d(x[d]; \alpha^{(d)})$ be a calibration function that acts on the d-th component of x and has parameters $\alpha^{(d)}$. For continous features, assume that they are bounded. Use e_d as the standard unit basis vector for the d-th component. Then one can write $c(x; \alpha) = \sum_{d=1}^D e_d c_d(e_d^T x; \alpha^{(d)})$. Then the proposed calibrated monotonic lattice regression objective expands the monotonic lattice regression objective to:

$$\arg \min_{\theta} \sum_{i=1}^n \mathcal l(y_i, \theta^T \phi(c(x_i; \alpha))) + R(\theta), \quad s.t. \quad A\theta \le b \quad \textrm{and} \quad \tilde A \alpha \le \tilde b,$$

where each row of A specifies a monotonicity constraint for a pair of adjacent lattice parameters and each row of $\tilde A$ similarly specifies a monotonicity constraint for a pair of adjacent calibration parameters for one of the piecewise linear calibration functions.

This turns the convex optimization problem into a non-convex optimization problem. Can be efficiently solved using *projected SGD* (after taking a SGD step, the parameters may violated the constraints → projection onto the constraints fixes this)

9.4. Large-Scale Projection Handling

Standard projected SGD projects the parameters onto teh constraints after each SGD update. A full projection after each iteration is impractical and un-neccessary → two new strategies

9.4.1. Suboptimal Projection

For each new stochastic subgradient $\eta \Delta$, we create a set of active constraints initialized to 0, and, starting from the last parameter values, move along the portion of $\eta \Delta$ that is orthogonal to the current active set until we encounter a constraint, add this constraint to the active set, and then continue unit the update $\eta \Delta$ is exhausted or it is not possible to move orthogonal to the current active set. At all times, the parameters satisfy the constraints. Can be particulary fast because it's possible to exploit the sparisty of the monotonicity constraints and the sparsity of Δ. This strategy is sub-optimal, the parameters can "get stuck" at a corner of the feasible set, as illustrated in the following figure. In practice, the stochasticity eventually jiggles the parameters free.

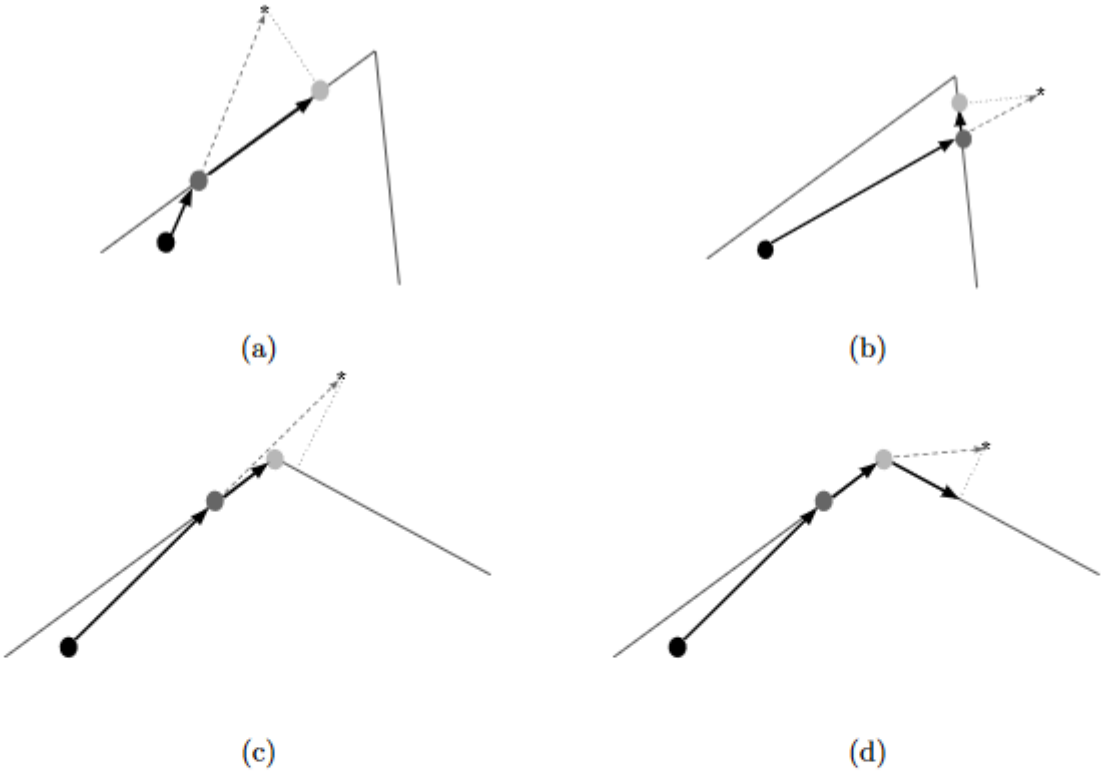


Figure 11: Four examples of the suboptimal projection stochastic gradient descent step described in Section 9.4. In each case, the constraints are marked by thin solid lines, the black dot represents the parameters at the end of the last SGD iteration, and the new full stochastic gradient descent update is marked by the dashed line, ending in a star. The optimal projection of the star onto the constraints is marked by the dotted line. The stochastic gradient is followed until it hits a constraint, and then the component of the remaining gradient orthogonal to the active constraint is applied. The update ends at the light gray dot. In cases (a) and (b), the resulting light dot is the optimal projection of the star onto the constraints. But in case (c), first one constraint is hit, and then another constraint is hit, and the update gets stuck at the corner of the feasible set without being able to apply all of the stochastic gradient. The resulting light gray dot is *not* the projection of the star onto the constraints, hence the projection for this iteration is suboptimal. However, it is likely that a future stochastic gradient will jiggle the optimization loose, as pictured in (d), producing an update that is again an optimal projection of the latest stochastic gradient.

9.4.2. Stochastic Constraints with LightTouch

At each iteration, LightTouch moves the constraints into the objective, and applies a random subset of constraints at each iteration as stochastic gradient updates to the parameters, where the distribution over the

constraints is learned as the optimizaion proceeds to focus on constraints that are more likely to be active. Intermediate solutions may not satisfy all the constraints, but one full projection is performed at the very end ⇒ optimal approach.

9.4.3. Adapting Stepsizes with ADAGRAD

Adagrad decays the step-size adaptively for each parameter, so that parameters updated more often or with larger magnitude gradients have a smaller step size.

10. Case Studies

11. Discussion and some Open Questions

Practical experience has shown, being able to check and ensure monotonocity helps users trust the model, and leads to models that generalize better. In the absence of prior information, use the direction of a linear fit to specify a monotonic direction and then use monotonicity as a regularizer.

First surprise: A simple 2^D lattice is often sufficient to capture the interactions of D features

Second surprise: Simplex interpolation provides similar accuracy as multilinear interpolation

One of the biggest speedups comes from randomly sampling the additive terms of the graph regularizers, analogous to the random sampling of the additive terms of the empirical liss that SGD uses.

The largest computational bottleneck remains the projection onto the monotonicity constraints.