Towards a unified view of trees, neural nets, and kernels at LinkedIn

Kinjal Basu

Sr. Staff Software Engineer

Ping Li

Distinguished Engineer

Mission

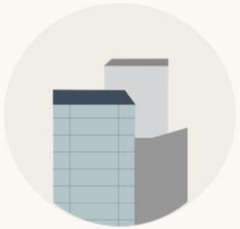
Connect the world's professionals
to make them more productive
and successful

LinkedIn’s Economic Graph

A digital representation of the global economy.

875M

Members



59M

Companies

129K

Schools

39K

Skills

Key Product offerings at LinkedIn

Recruiter

Help recruiters find suitable candidates

Jobs

Help job seekers find a good match

People You May Know

Connect members with other members they might know

Feed

Help members develop an active professional community

Few Major Challenging Problems

Across our different product offerings

Optimization

- Building complex models towards optimizing member value
- Multi-objective optimization
- Automatic parameter selection (Bayesian Optimization)
- Extreme-scale Linear optimization

Experimentation

- Speed of experimentation
  - Noise reduction
  - Adaptive Experimentation
  - Explore-Exploit
- Creator side experimentation
- A/B experiments under network interference

Responsible Design

- Fairness
  - Metric Definition
  - Large-scale mitigation
  - Large-causal models to explain unfairness
- Interpretability
  - Replacing black-box models
  - Enhancing Trust
- Privacy
  - Differential Privacy
  - Federated Learning

It takes a village

- AI + Data Science
- Engineering
- Product
- Legal
- Comms
- Ethics
- Policy

Towards a unified view of trees, neural nets, and kernels

(Examples of Research Topics at LinkedIn)

- Boosted trees have achieved state-of-the-art performance in numerous machine learning tasks and widely used in practice.
- Neural nets have been dominantly popular in the past decade.
- Kernels are perhaps not much the focus of machine learning community and practice these days.
- We work on research in all three areas with the hope to ultimately achieve a unified view of trees, neural nets, and kernels.

Let's begin with a ridge regression problem

http://www.cs.toronto.edu/~delve/data/datasets.html

"comp-cpu" (CPU) regression dataset: 21 features, 8192 examples (50%/50% for training/testing)

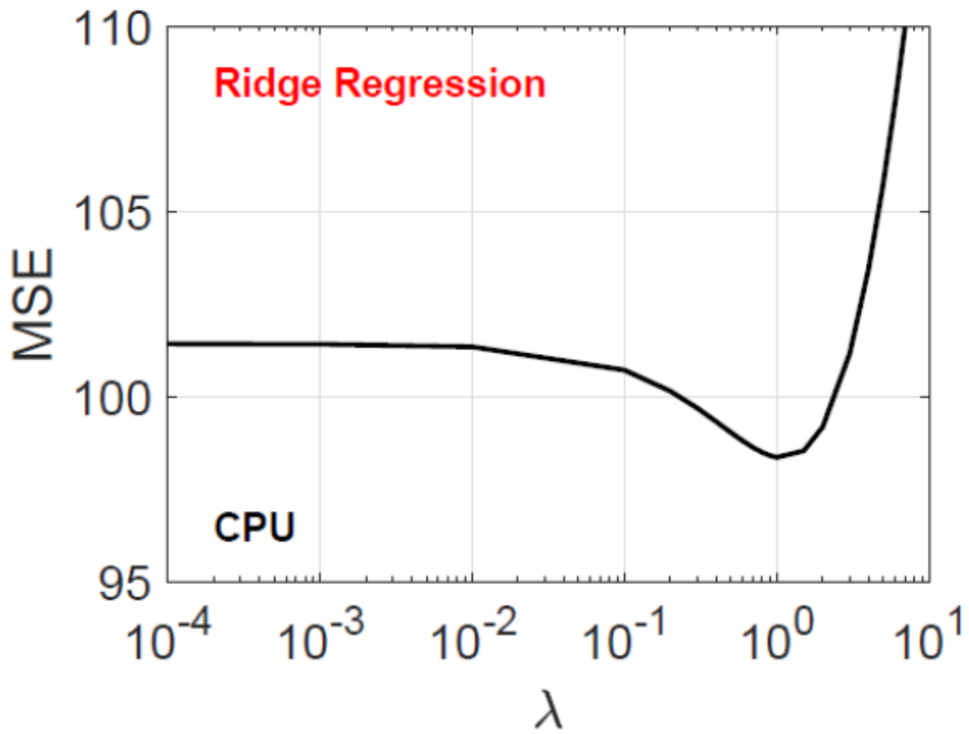
Ridge Linear Regression (LR) $\hat{Y}_t = X_t\left(X^T X + \lambda I_p\right)^{-1} X^T Y, \quad X \in R^{n \times p}$

Y, X : training data

Y_t, X_t: testing data

Test mean square error (MSE)

$$MSE_t = \frac{1}{n_t}\sum\left(\hat{Y}_t - Y_t\right)^2$$



Let's begin with a ridge regression problem

Ridge Regression $\hat{Y}_t = X_t\left(X^TX + \lambda I_p\right)^{-1}X^TY, \quad X \in R^{n\times p}$

The trick of "Push−Through Identity" $\left(X^TX + \lambda I_p\right)^{-1}X^T = X^T\left(XX^T + \lambda I_n\right)^{-1}$

Rewrite Ridge Regression $\hat{Y}_t = X_tX^T\left(XX^T + \lambda I_n\right)^{-1}Y$

Linear kernels $X_tX^T, \quad XX^T$

From ridge regression to kernel regression

$$\hat{Y}_t = X_t X^T \left(XX^T + \lambda I\right)^{-1} Y = K_t (K + \lambda I)^{-1} Y, \qquad K_t = X_t X^T, \qquad K = XX^T$$

More generally, we can use other kernels such as the Gaussian (RBF) Kernel, or

Min-Max Kernel $\qquad K(u, v) = \frac{\sum \min(u_i, v_i)}{\sum \max(u_i, v_i)}$, for two data vectors u, v

This simple (and perhaps unusual) kernel has no tuning parameters!

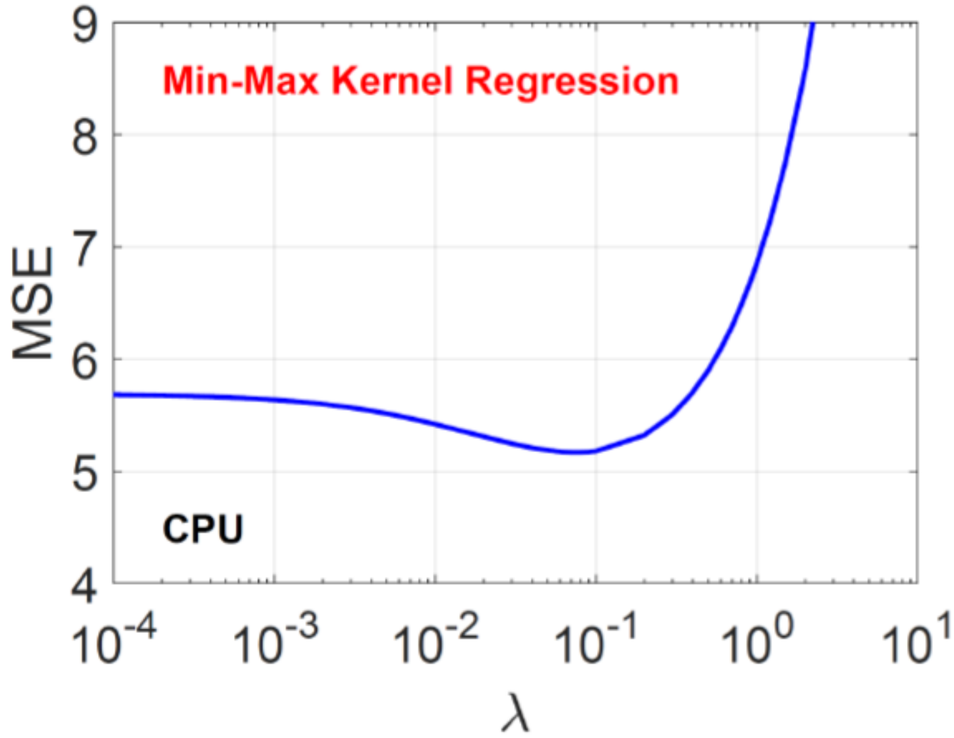
Min-max kernel ridge regression

$$\hat{Y}_t = K_t(K + \lambda I)^{-1}Y$$

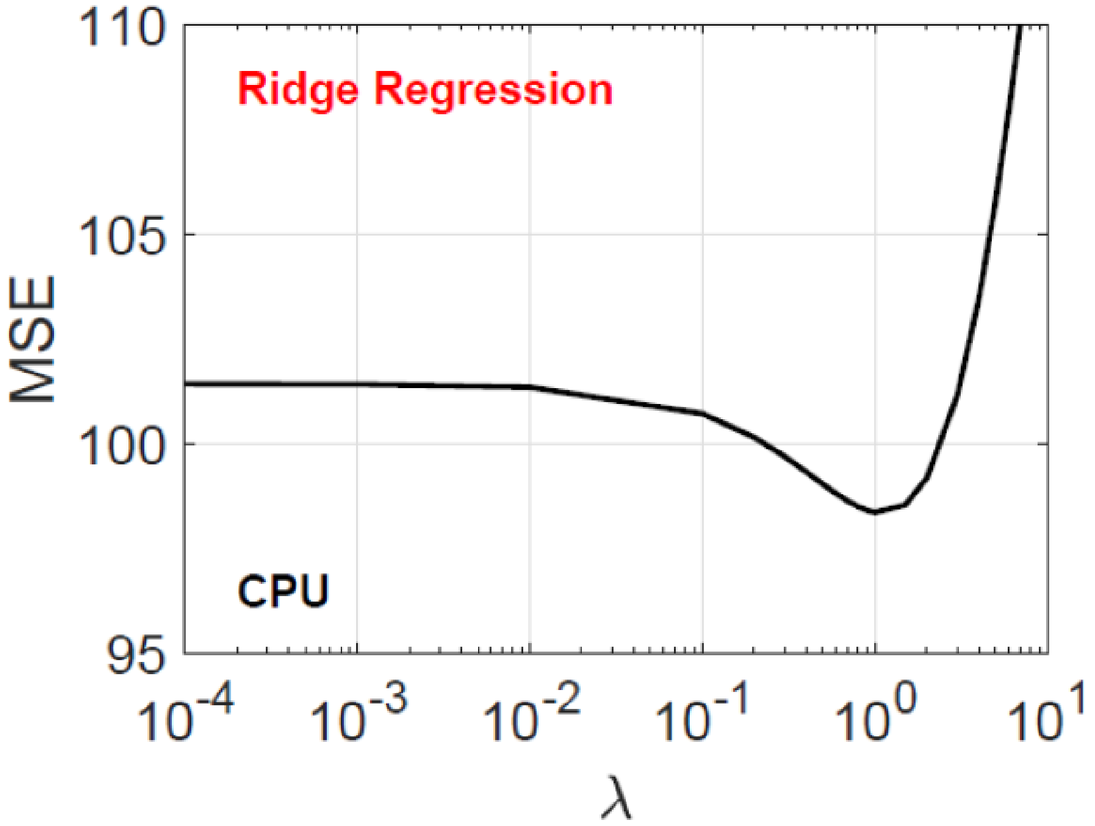
where K_t and K, are min-max kernels and computed from data

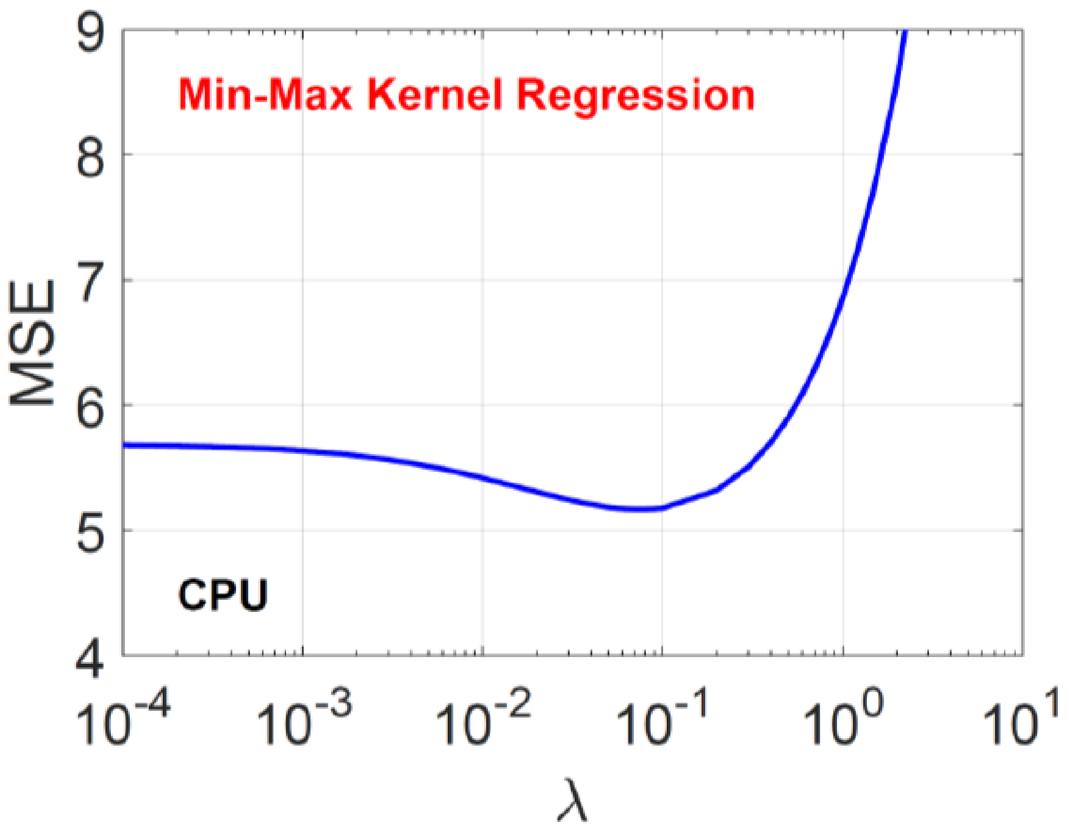
Min-Max Kernel $K(u,v) = \frac{\sum \min(u_i, v_i)}{\sum \max(u_i, v_i)}$, for two data vectors u, v

Test mean square error (MSE)



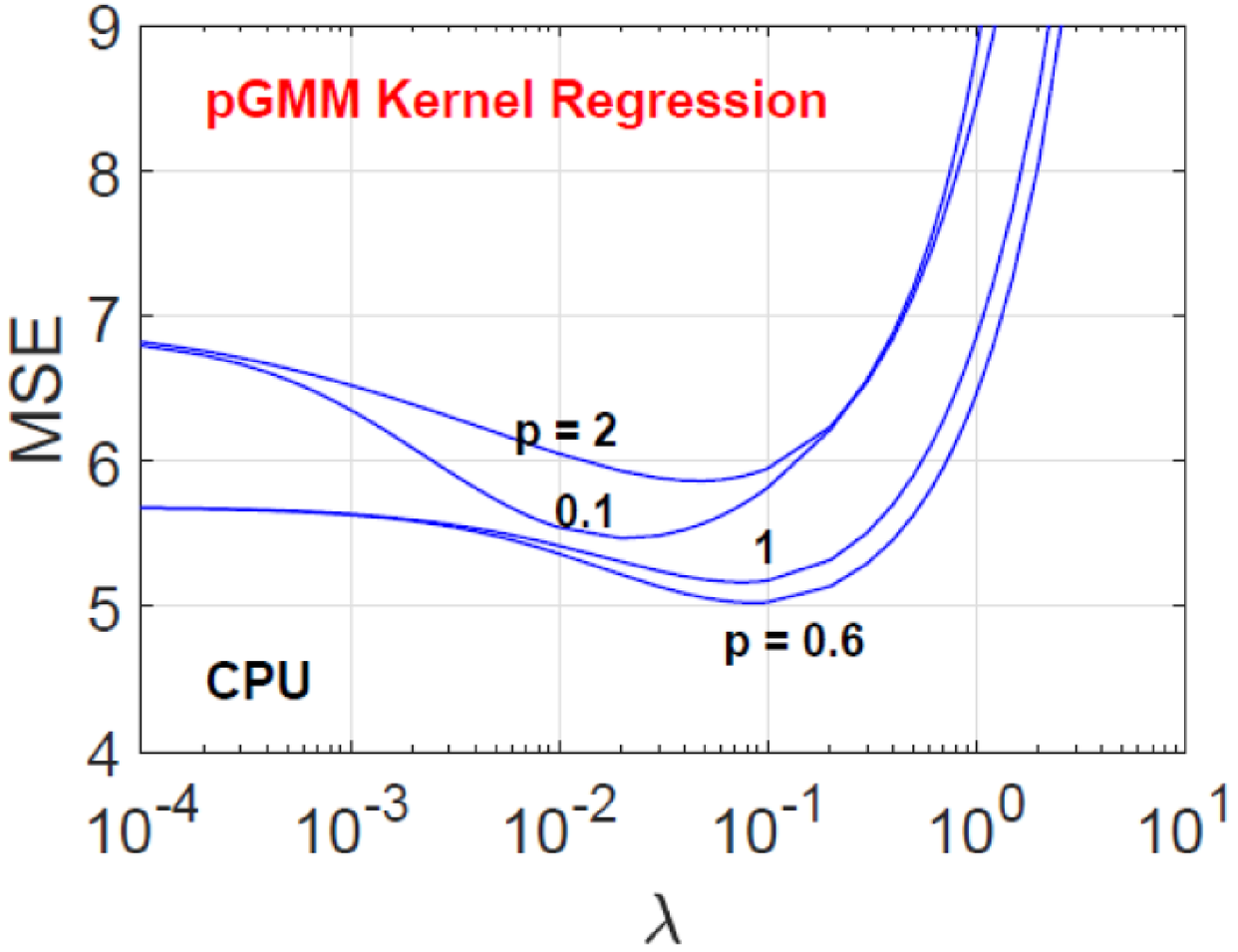
Comparing ridge with min-max kernel regression





Using (tuning-free) Min-Max Kernel, test MSEs are substantially reduced, without tuning parameter

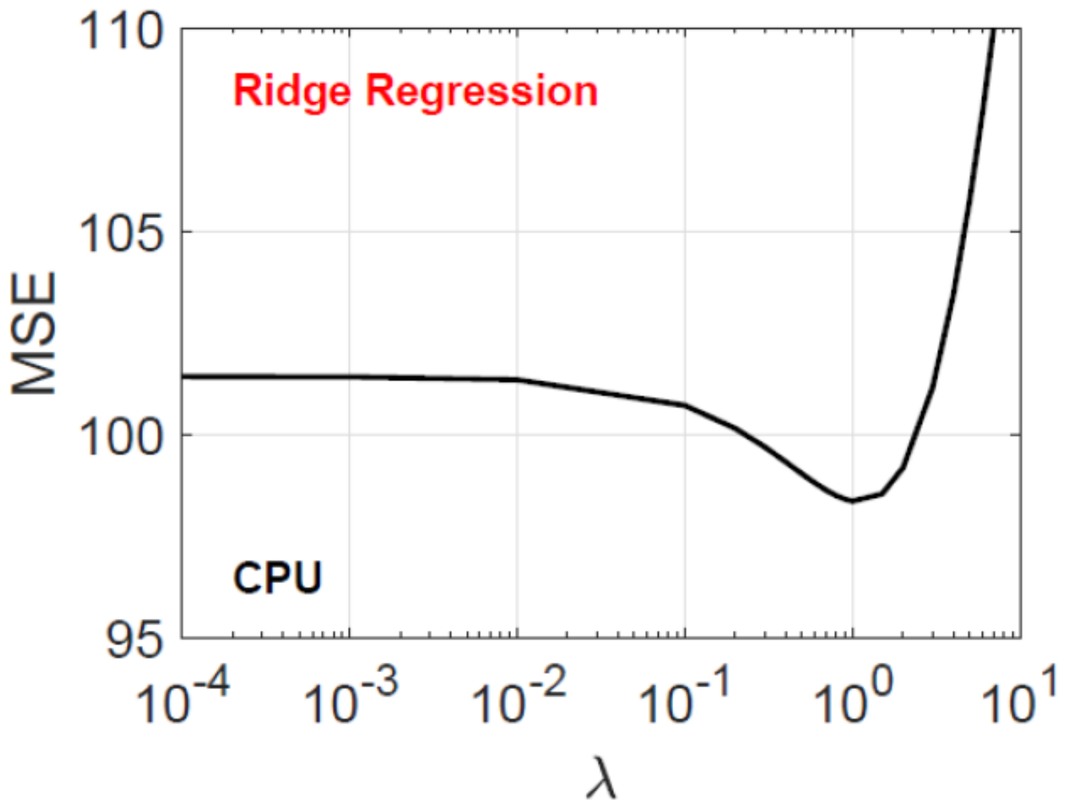
Powered generalized min-max (pGMM) kernel regression

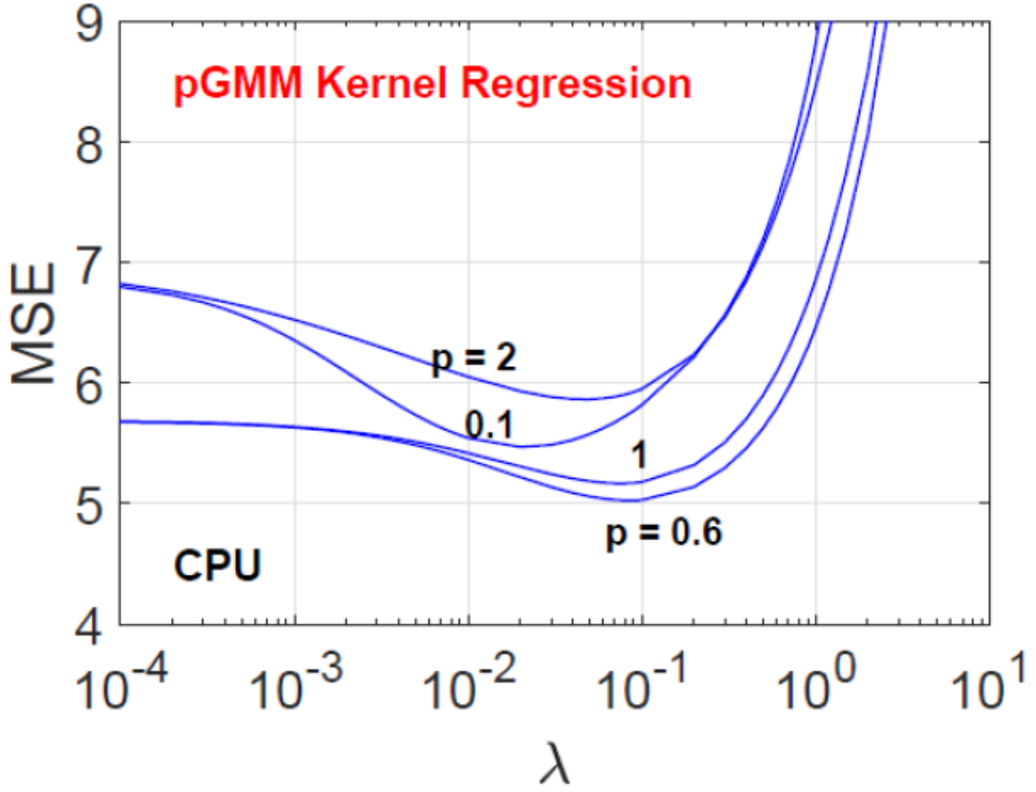


$$K(u,v;p) = \frac{\sum \min(u_i, v_i)^p}{\sum \max(u_i, v_i)^p}$$

It turns out that p = 1 is not the best choice

The pGMM kernel regression





When p = 0.6, test MSE can be as small as 5.03, compared to 98.35 in original ridge regression.

(**L2 boosted tree** model can also achieve test MSE around 4.69)

Comparisons with Gaussian kernel and L2-Boosting

Table 1: Datasets for testing regression algorithms. We report the best test mean square error (MSE) for each method, over the range of regularization coefficients and parameters.

dataset	# train	# test	dim	LR	RBF	GMM	pGMM	L_2-Boost
ENBcool	384	384	8	10.24	3.20	1.70	1.28	1.21
ENBheat	384	384	8	9.00	0.495	0.191	0.188	0.186
Airfoil	752	751	5	24.26	8.35	7.50	3.56	3.09
CPUsmall	4096	4096	12	102.12	9.05	7.22	7.05	6.89
CPU	4096	4096	21	98.35	6.42	5.17	5.03	4.69
WECPerth	5000	5000	32	1.1×10^9	2.3×10^8	2.4×10^8	2.3×10^8	2.5×10^8
Cadata	10320	10320	8	4.8×10^9	3.8×10^9	2.4×10^9	2.4×10^9	2.1×10^9
House16H	11392	11392	16	2.1×10^9	1.3×10^9	1.2×10^9	1.1×10^9	1.0×10^9
House16L	11392	11392	16	1.9×10^9	1.1×10^9	9.9×10^8	9.4×10^8	8.6×10^8
CASP	22865	22865	9	26.63	23.94	15.30	15.29	13.51
Splice	1000	2175	60	0.1205	0.0967	0.0589	0.0589	0.0352
Mnoise1	2519	454	784	0.0484	0.0344	0.0311	0.0169	0.0145
Mnoise6	2519	454	784	0.0215	0.0165	0.0195	0.0129	0.0131
Mimage	2538	10524	784	0.0540	0.0323	0.0263	0.0125	0.0149

Kernels, trees, and neural nets

- The (tuning-free) GMM kernel is surprisingly effective, compared to the best-tuned Gaussian kernels. Tunable GMM kernels, e.g., the pGMM, in some cases can lead to even substantially better results than (tuning-free) GMM.

- The pGMM kernel often performs similarly to boosted trees.

- The pGMM kernel is nonlinear and will need to be linearized before this kernel can be used for large scale problems. Hashing is a convenient strategy.

- Hashed data for the pGMM kernel can also be input to the deep neural nets.

Recent developments on boosting and trees

Classical works on boosting and trees before early 2000 were developed by pioneers including Schapire, Freund, Bartlett, Singer, Friedman, Hastie, Tibshirani, etc.

More recently, three major developments have made boosted trees more practical and more accurate.

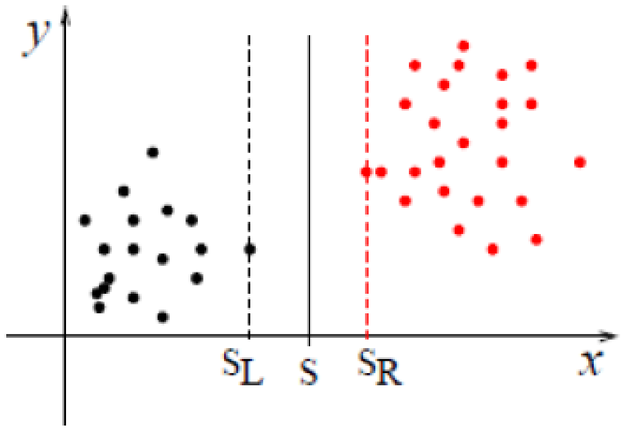
1. The adaptive binning strategy for effectively transforming any data types into integers. This makes tree implementations much more convenient and more efficient. [Ping Li et al, NIPS 2007]

2. The gain formula for tree-split using 2^{nd}-order information. This (in retrospect) simple formula has often made trees substantially more accurate. Using only 1^{st}-order information sometimes did not beat kernel SVMs. This formula is now the standard implementation of popular tree platforms. [Ping Li, UAI 2010]

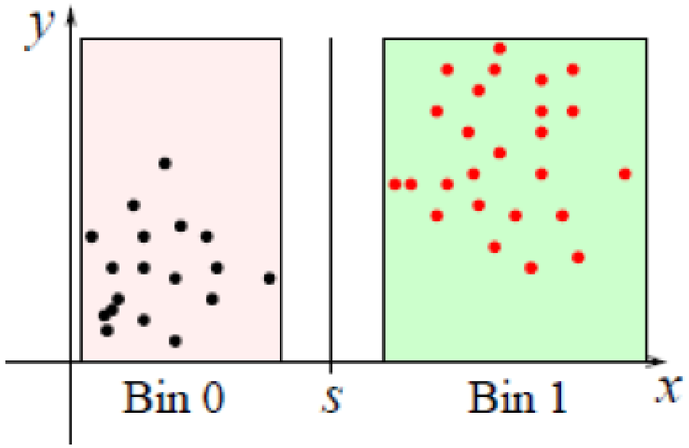
3. The new derivatives (different from textbooks) of logistic regression by assuming a base class and strategies for selecting the base class. These have improved the accuracy of many multi-class classification tasks. [Ping Li, ICML 2009]

The adaptive feature binning strategy

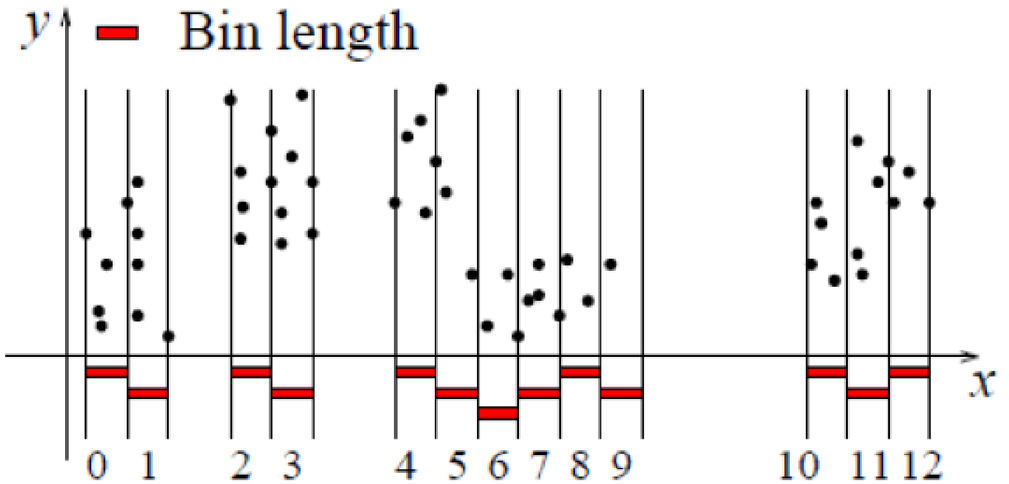
NIPS 2017: McRank: Learning to Rank Using Multiple Classification and Gradient Boosting



Tree algorithm only splits where there are data and makes arbituary decision where they are no data



For this split, two bins (0,1) migth be sufficient.



Therefore, we can divide the each dimention (feature) into equal-lengh bins (for simplisity) but we only assign bins where there are data. This strategy is simple and effective.

The gain formula for tree-split using 2nd-order information

UAI 2010: Robust LogitBoost and Adaptive Base Class (ABC) LogitBoost

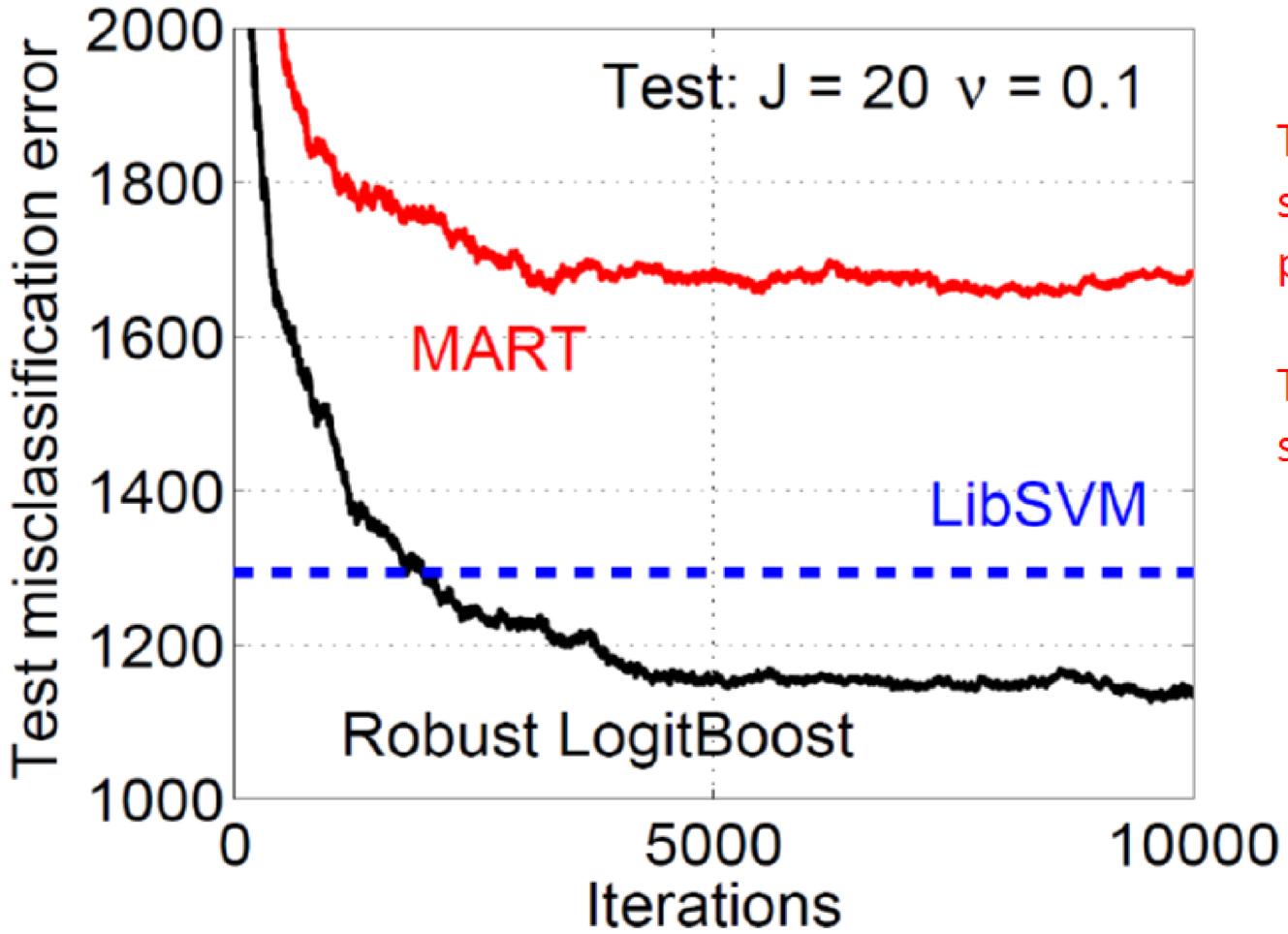
This formula is behind the success of popular tree platforms:

$$Gain(t) = \frac{\left[\sum_{i=1}^{t}\left(r_{i,k}-p_{i,k}\right)\right]^2}{\sum_{i=1}^{t} p_{i,k}(1-p_{i,k})} + \frac{\left[\sum_{i=t+1}^{N}\left(r_{i,k}-p_{i,k}\right)\right]^2}{\sum_{i=t+1}^{N} p_{i,k}(1-p_{i,k})} - \frac{\left[\sum_{i=1}^{N}\left(r_{i,k}-p_{i,k}\right)\right]^2}{\sum_{i=1}^{N} p_{i,k}(1-p_{i,k})}.$$

The gain formula for tree-split using 2nd-order information

UAI 2010: Robust LogitBoost and Adaptive Base Class (ABC) LogitBoost

This is the result from the 2010 UAI paper (in Appendix), which might be very slghtly different from the output of the current package.



This largely explains the success of boosting in practice in past decade:

The second-order tree split formula is the key.

Lp boosting using 2nd-order information

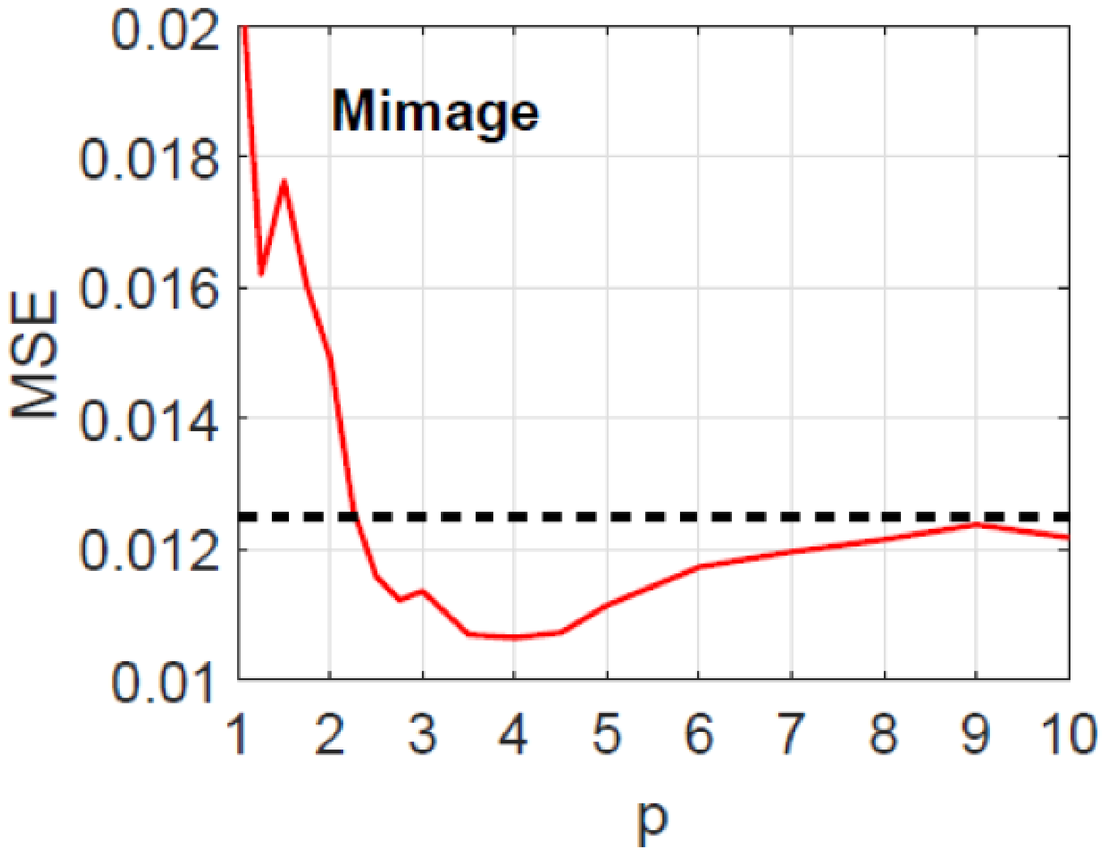
$$L_p = \frac{1}{n}\sum_{i=1}^{n} L_{p,i} = \frac{1}{n}\sum_{i=1}^{n} |y_i - F_i|^p, \qquad \text{where } F_i = F(\mathbf{x}_i).$$

$$\frac{\partial L_{p,i}}{\partial F_i} = -p|y_i - F_i|^{p-1}\text{sign}\,(y_i - F_i)$$

$$\frac{\partial^2 L_{p,i}}{\partial F_i^2} = p(p-1)|y_i - F_i|^{p-2}.$$ = 2 if p = 2 (the usual L2 boosting)

$$Gain(s) = \frac{[\sum_{i=1}^{s} L'_i]^2}{\sum_{i=1}^{s} L''_i} + \frac{\left[\sum_{i=s+1}^{N} L'_i\right]^2}{\sum_{i=s+1}^{N} L''_i} - \frac{\left[\sum_{i=1}^{N} L'_i\right]^2}{\sum_{i=1}^{N} L''_i}.$$

Lp boosting using 2nd-order information



For some datasets, the difference between L2 boost and Lp boost can be substantial

https://arxiv.org/pdf/2207.08667.pdf

ABC-Boost: adaptive base class boost for multi-class

ICML 2009: ABC-Boost: Adaptive Base Class Boost for Multi-class Classification

In textbooks, these are the first and second derivatives of logistic regression loss function:

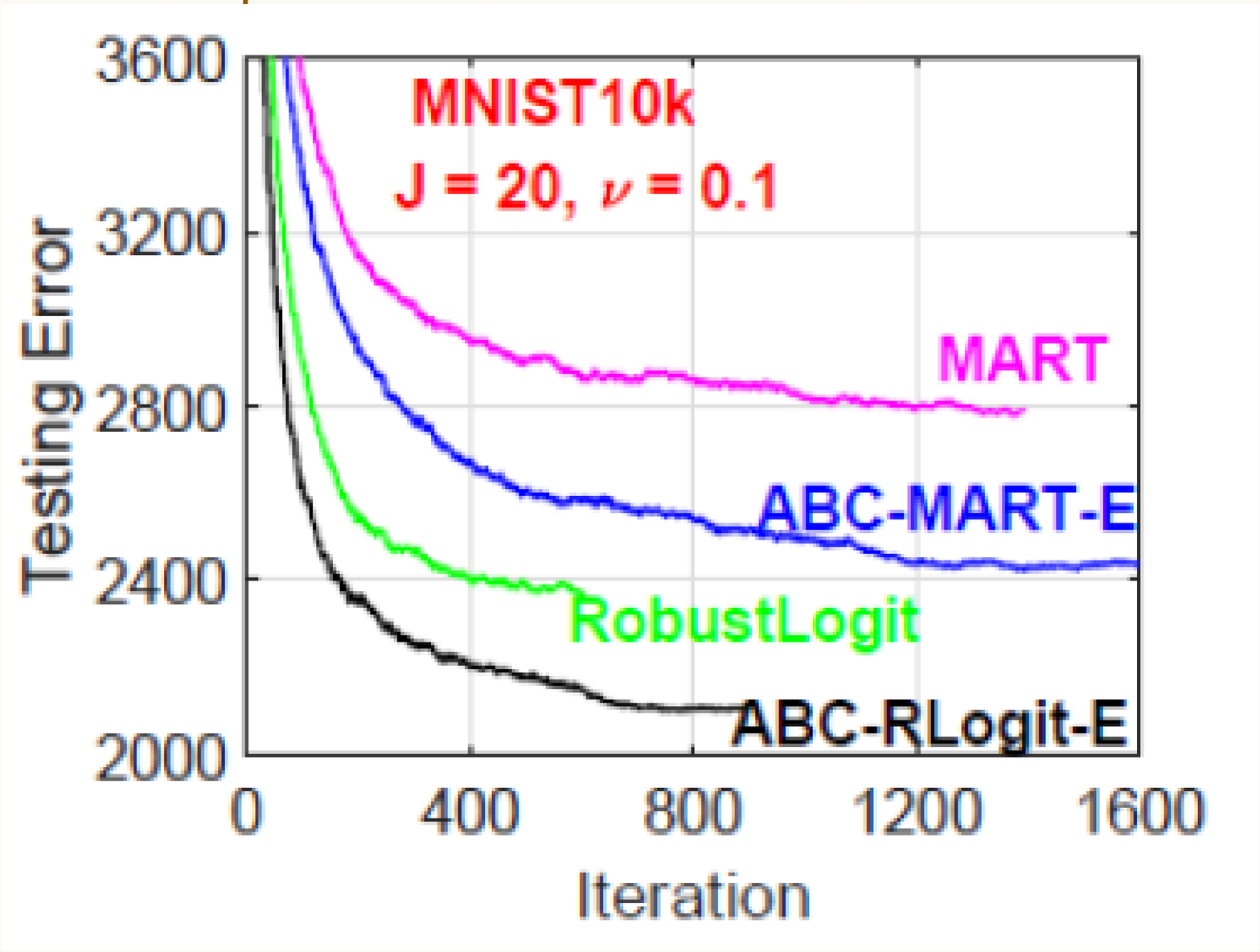
$$\frac{\partial L_i}{\partial F_{i,k}} = -\left(r_{i,k} - p_{i,k}\right), \qquad \frac{\partial^2 L_i}{\partial F_{i,k}^2} = p_{i,k}\left(1 - p_{i,k}\right).$$

The ICML 2009 paper derived a new set of derivatives, by assuming class 0 is "base class":

$$\frac{\partial L_i}{\partial F_{i,k}} = \left(r_{i,0} - p_{i,0}\right) - \left(r_{i,k} - p_{i,k}\right),$$

$$\frac{\partial^2 L_i}{\partial F_{i.k}^2} = p_{i,0}(1 - p_{i,0}) + p_{i,k}(1 - p_{i,k}) + 2p_{i,0}p_{i,k}.$$

ABC-Boost: adaptive base class boost for multi-class

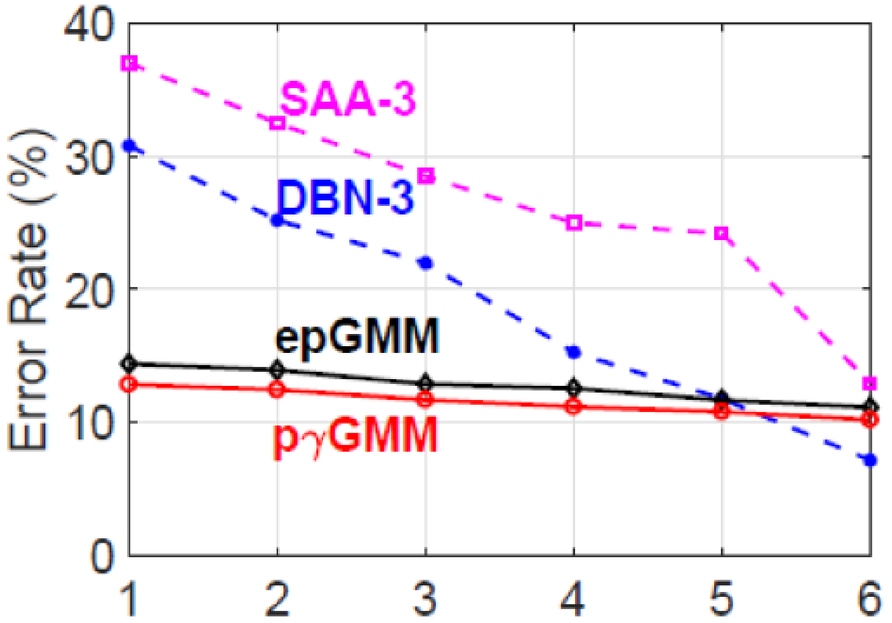


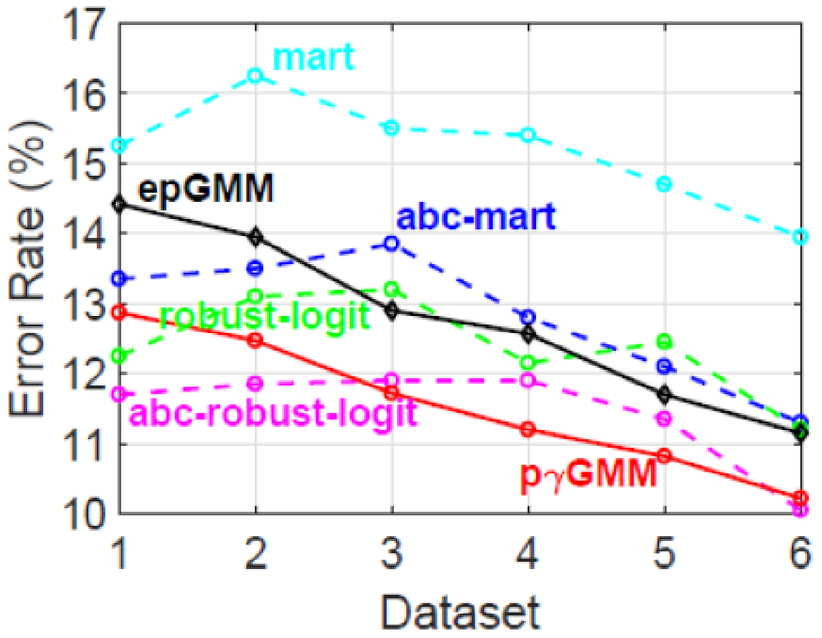
Comparing kernels, boosted trees, and neural nets

https://arxiv.org/pdf/1701.02046.pdf
https://arxiv.org/pdf/1805.02830.pdf

Six additional datasets created by CIFAR for deep learning study

Error rate, lower the better





Tunable Min-Max Kernels do better than deep nets in most cases.

Tunable Min-Max Kernel can be close to boosted tree models

GCWSNet: hashing the pGMM kernel for neural nets

$$GMM(u,v) = \frac{\sum_{i=1}^{2D} \min\{\tilde{u}_i, \tilde{v}_i\}}{\sum_{i=1}^{2D} \max\{\tilde{u}_i, \tilde{v}_i\}}$$

$$pGMM(u,v;p) = \frac{\sum_{i=1}^{2D} (\min\{\tilde{u}_i, \tilde{v}_i\})^p}{\sum_{i=1}^{2D} (\max\{\tilde{u}_i, \tilde{v}_i\})^p}$$

$$\begin{cases} \tilde{u}_{2i-1} = u_i, \quad \tilde{u}_{2i} = 0 & \text{if } u_i > 0 \\ \tilde{u}_{2i-1} = 0, \quad \tilde{u}_{2i} = -u_i & \text{if } u_i \leq 0 \end{cases} \tag{1}$$

GCWSNet: hashing the pGMM kernel for neural nets

Dataset	# train, # test, # dim	linear	RBF	GMM	pGMM (p)
SEMG	1800, 1800, 2500	19.3	29.0	54.0	56.1 (2)
DailySports	4560, 4560, 5625	77.7	97.6	99.6	99.6 (0.6)
M-Noise1	10000, 4000, 784	60.3	66.8	71.4	85.2 (80)
M-Image	12000, 50000, 784	70.7	77.8	80.9	89.5 (50)
PAMAP101	188209, 188208, 51	75.3	—	—	—
Covtype	290506, 290506, 54	71.5	—	—	—

The pGMM kernel achieves good accuracy.

However, the pGMM kernel is nonlinear and cannot be easily scaled up for datasets of moderate sizes.

GCWSNet: hashing the pGMM kernel for neural nets

This is a numerically stable power-transformation on data.

Otherwise, applying power-transformations can be tricky.

Algorithm 1 Generalized consistent weighted sampling (GCWS) for hashing the pGMM kernel.

Input: Data vector u_i ($i = 1$ to D)

Generate vector $\tilde{u}$ in $2D$-dim by (1).
for i from 1 to $2D$ **do**
$r_i \sim Gamma(2,1)$, $c_i \sim Gamma(2,1)$, $\beta_i \sim Uniform(0,1)$
$t_i \leftarrow \lfloor p \frac{\log \tilde{u}_i}{r_i} + \beta_i \rfloor$, $a_i \leftarrow \log(c_i) - r_i(t_i + 1 - \beta_i)$
end for

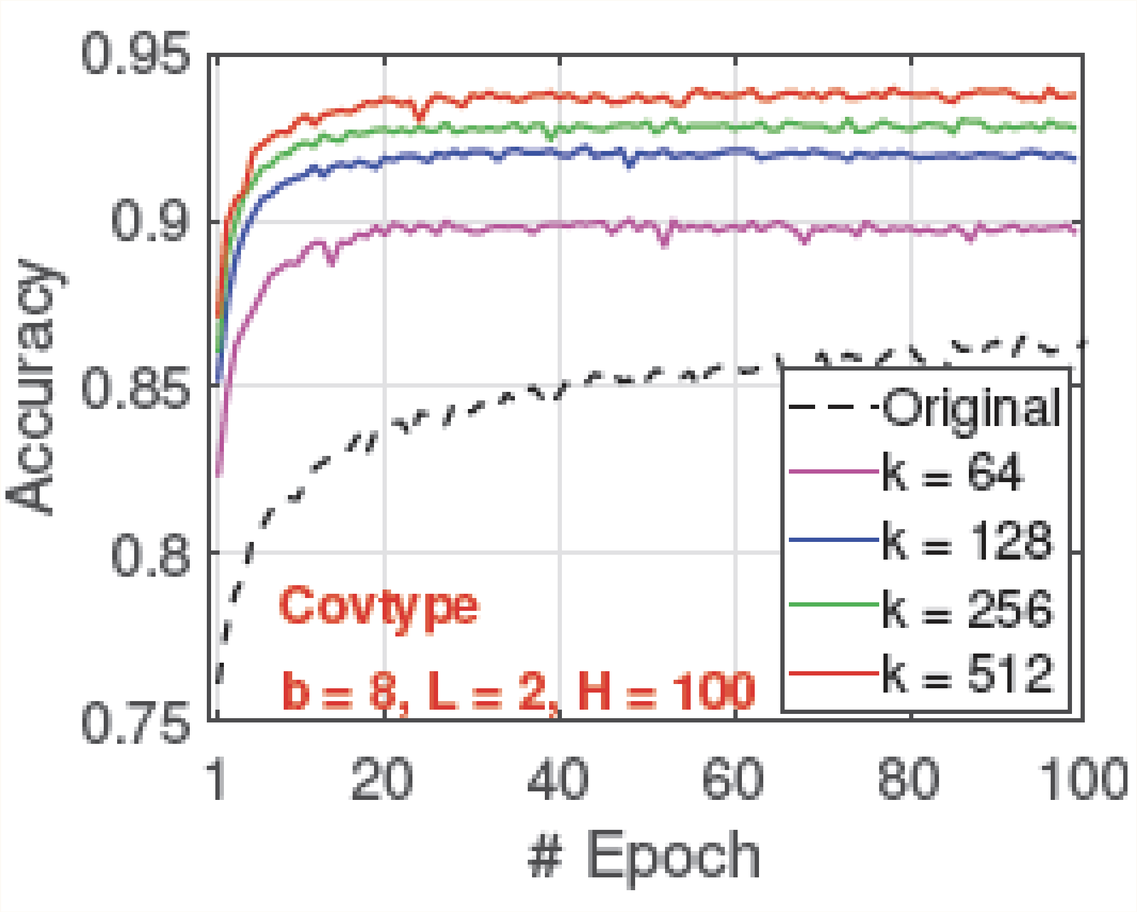
Output: $i^* \leftarrow arg\min_i \ a_i$, $t^* \leftarrow t_{i^*}$

$$P[(i_u^*, t_u^*) = (i_v^*, t_v^*)] = pGMM(u, v)$$

This means that we can view integers i's as categorical variables and use them (in one-hot representation) as input to approximate the pGMM kernel.

$$P[i_u^* = i_v^*] \approx P[(i_u^*, t_u^*) = (i_v^*, t_v^*)] = pGMM(u, v)$$

GCWSNet: hashing the pGMM kernel for neural nets

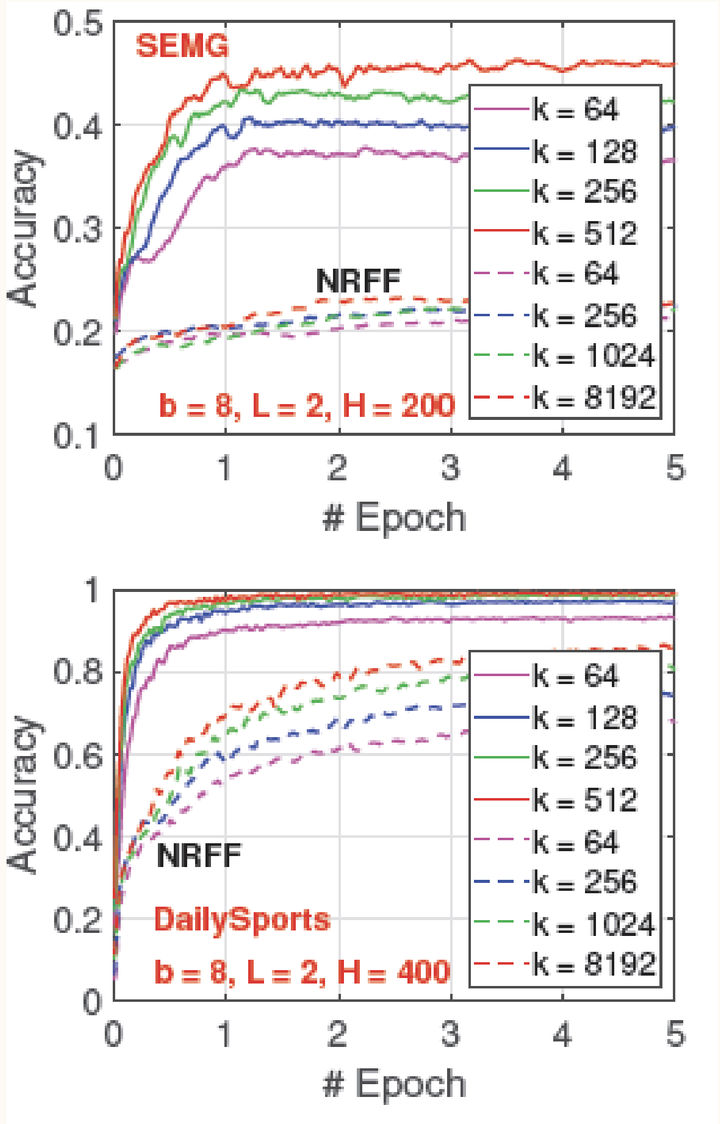


Test accuracy for neural nets for 1 hidden layer (H=100 nodes) .

Dashed lines are the results of using original data.

Solid lines correspond to GCWSNet with different (k) hashes.

GCWSNet: hashing the pGMM kernel for neural nets



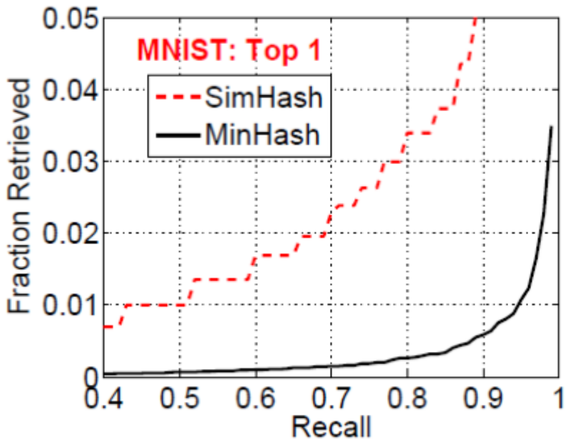
More experiments and comparisons with normalized random Fourier features (NRFF, dashed lines)

RFF is a popular hashing method. NRFF improves RFF via normalization (Li, P. KDD 2017)

Embedding based retrieval (EBR) with GCWS

For binary data, GCWS is equivalent to "minwise hashing" (or minhash), which is typically a better indexing scheme than the popular "sign random projections" (or simhash) for approximate near neighbor search.

In Defense of MinHash Over SimHash, AISTATS 2014



Fraction retrieved versus recall plots:
standard way to evaluate ANN algorithms

Exhaustive search = 100% fraction retrieved

Recall = percentage of ground truths included in search results

SimHash: In order to achieve a recall at 90% (0.9), we need to search for 5% of the data points. 5% means a 20-fold reduction in cost, pretty good.

MinHash: In order to achieve a recall at 90% (0.9), we need to search for 0.5% of the data points. 0.5% is ten times better than 5%.

Like minwise hashing, GCWS generates integer hashes, which can be conveniently used for indexing.

Embedding based retrieval (EBR) research at LinkedIn

Many research (and deployment) activities on embedding based retrieval (EBR) are on-going at LinkedIn.

SignRFF: Sign Random Fourier Features

Xiaoyun Li, Ping Li
LinkedIn Corporation
700 Bellevue WA NE, Bellevue, WA 98004, USA
{lixiaoyun996,pingli98}@gmail.com

To appear in NeurIPS 2022

https://openreview.net/pdf?id=ZfaEZyQDrok

Embedding-Based Retrieval (EBR) and Beyond

approximate near neighbor (ANN), similarity estimation, hashing techniques, compact representations

Ping Li, LinkedIn
https://www.linkedin.com/in/ping-li-a4624389/

700 Bellevue Way NE, Bellevue, WA 98004, USA

August 30, 2022

A recent talk on EBR

https://www.linkedin.com/feed/update/urn:li:activity:6972533869245865984/

Towards a unified view of trees, neural nets, and kernels

There have been many research activities on boosted trees, neural nets, kernels, at LinkedIn.

In terms of accuracy, on tasks with given features, boosted trees typically achieve the best accuracy, with recent efforts including feature binning, 2nd-order tree-split, abc-boost, etc.

Neural nets and hashed kernels are efficient, and the accuracy can also be good if used properly. For industrial advertising models with 1000 billion features (~1000 categorical features with many levels), neural nets are currently the standard tool for production.

The pGMM kernel performs surprisingly well in terms of accuracy, it is also efficient via hashing by the generalized consistent weighted sampling (GCWS). The theory of the pGMM kernel is rare (e.g., "Theory of the GMM kernel", 2017) and might be urgently needed.

Reach out!

Email :

kbasu@linkedin.com

pinli@linkedin.com

We'd love to hear from you!

- If you are planning to use one of the LinkedIn tools.
- Have an idea in AI and want to explore further.
- Or to simply say ...

Thank You!

Questions??