Task-driven Greedy Learning of Feature Hashing Functions

Artem Sokolov Stefan Riezler
Heidelberg University, Germany
{sokolov, riezler}@cl.uni-heidelberg.de

Classical Random Feature Hashing [Ganchev and Dredze, 2008, Shi et al., 2009, Weinberger et al., 2009]

Consider **classification task** (for simplicity):

- ➡ sparse **high-dimensional** labeled data: $(\phi_n, y_n) \in \mathbb{R}^D \times \{-1,+1\}, D \gg 1$
- ➡ **linear** scoring $f(\phi; \mathbf{w}) = \langle \mathbf{w}, \phi \rangle$
- ✔ high dimension $D \Rightarrow$ approximate separability
- ✖ **But we need to access w rapidly:**
  - ➡ must keep $\mathbf{w}$ in RAM $\rightarrow$ may not fit
  - ➡ storage model matters:
    - (un)ordered associative tables need RAM and/or are slow
    - ✔ **linear arrays are fastest** $\Rightarrow$ **need integer feature indexes**

Common solution:

- **feature hashing trick** (alphabet elimination/random feature mixing)
- **Idea:**
  - per-coordinate mapping into a lower-dimension feature space
  - with **data-independent** pseudo-random function $HASH$:

$$\phi'_{d'} = \sum_{d:HASH(d)=d'} \phi_d, \qquad \text{where } d \text{ is a feature key (usually a string)}$$

- ✔ $||\phi'|| \simeq ||\phi||$ with high probability
- ✔ works surprisingly well: **little or no sacrifice in quality**!
- ✔ implemented in many learning kits (Vowpal Wabbit, etc.)

Can we do even better with **data-dependant** hashing ?

Paper Highlights

- ✔ replaces random feature hashes with **learned feature hashes**
- ✔ **simple greedy algorithm** for learning hashes
- ✔ learning **optimizes task objective**
- ✔ data-dependent hashing **improves classification** accuracy:
  - ★ for high-dimensional features
  - ★ in case of tight memory constrains (many collisions)

New hash function $HASH_{new}$ that:

- is informed of the final learning task
- can be done in preprocessing
- leverages existing optimization procedures

Idea Overview

Intuition: hashing d, d' together entails less loss change if $w_d \simeq w_{d'}$
$\Rightarrow$ approximate $w_d - w_{d'}$ as dist. between learnable representations:

0 Equip each feature d with a vector $\boldsymbol{\nu}(d) \in \mathbb{R}^V$ (e.g. some NLP stats about d's usage in the wild)

1 Learn a map $\boldsymbol{\nu}(d) \mapsto \mathcal{H}(\boldsymbol{\nu}(d)) \in \{0,1\}^T$, s.t. the Hamming dist. between $\mathcal{H}(\boldsymbol{\nu}(d_1)), \mathcal{H}(\boldsymbol{\nu}(d_2))$ captures d_1, d_2's similarity for task

2 Apply surjection $\mathcal{B} : \{0,1\}^T \rightarrow \{0\dots M\}$, s.t. close Hamming vectors get projected into the same integer

3 Define $HASH_{new} = \mathcal{H} \circ \mathcal{B}$, interpret outputs as RAM addresses.

Learning Feature Hashes by Optimizing Hinge Loss with Boosting

1: greedy learning of Hamming representation $\mathcal{H}$

- will look for w_d representable as $\sum_{t \le T} \alpha_t h_t(\boldsymbol{\nu}(d))$
- **close** $\mathcal{H}(\boldsymbol{\nu}(d)) = [h_1(\boldsymbol{\nu}(d)), \dots, h_T(\boldsymbol{\nu}(d))]$ for different $d \Rightarrow$ **close** values of respective w_d.

Hinge loss: $L = \sum_n \left(1 - y_n \sum_d \sum_{t \le T} \alpha_t h_t(\boldsymbol{\nu}(d)) \phi_{n,d}\right)_+$
Learning $\mathcal{H}(\boldsymbol{\nu}_d)$ in a **boosting** fashion:

- $\mathcal{H}^{t-1}(\boldsymbol{\nu}) = [h_1(\boldsymbol{\nu}), \dots, h_{t-1}(\boldsymbol{\nu})] \in \{0,1\}^{t-1}$
- $\mathcal{H}^t(\boldsymbol{\nu})$ is obtained by appending a bit-function – a per-coordinate decision stump $h_t = [\![\nu_{k^*} > \theta^*]\!]$:

$$k^*, \theta^* = \arg\max_{k,\theta} \left| \sum_{n: y_n \langle \mathbf{w}^t, \boldsymbol{\phi}_n \rangle < 1} y_n \sum_d \phi_{n,d} [\![\nu_{d,k} \ge \theta]\!] \right|$$

2: distance-sensitive projection $\mathcal{B}$

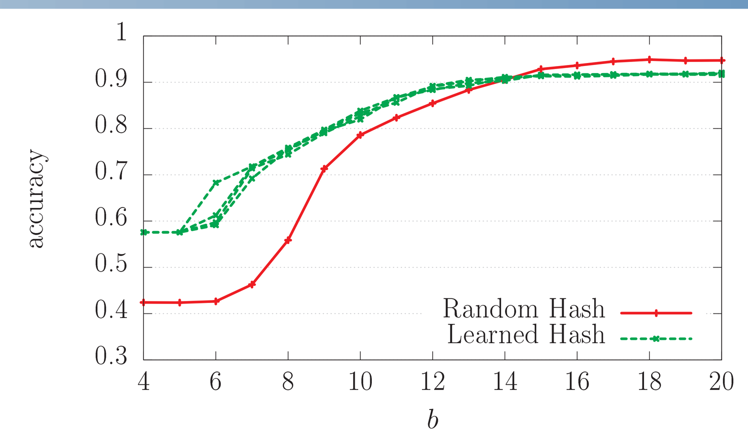
- **Task**: compress $\mathcal{H}(\boldsymbol{\nu})$ into short codes that have a high collision probability for close $\mathcal{H}(\boldsymbol{\nu})$.
- we use the **KOR random traces** [Kushilevitz et al., 1998]:
  - for a bit-vector $\boldsymbol{h} = [h_1, \dots, h_T]$
  - and random Bernoulli vectors $\boldsymbol{r}_m = [r_{m,1}, \dots, r_{m,T}]$
  - the trace is $\boldsymbol{t} = [t_1, \dots, t_M]$, s.t. $t_m = \langle \boldsymbol{h}, \boldsymbol{r}_m \rangle \bmod 2$.
- t_m has a bias towards closer $\boldsymbol{h}$, amplified by repeating M times
- **collision probability decays** with $D_H(\boldsymbol{h}_1, \boldsymbol{h}_2)$ and M:

$$P\left[\boldsymbol{t}_1 = \boldsymbol{t}_2 | D_H(\boldsymbol{h}_1, \boldsymbol{h}_2) \le \Delta\right] \ge \left(\frac{1}{2} + \frac{1}{2}(1-2p)^{\Delta}\right)^M$$

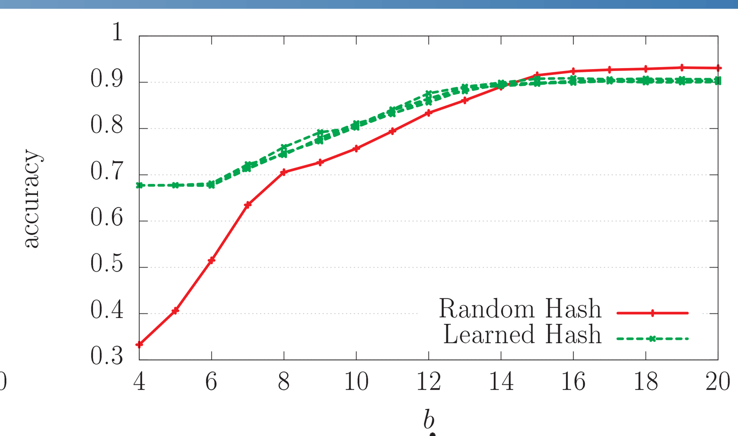
Proof-of-Concept Experiments

★ dataset #1: **20-newsgroups**

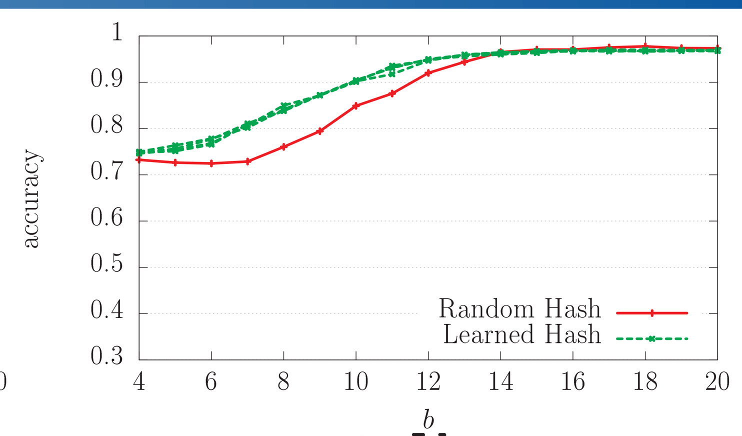
- 3 (one vs. all) class. tasks for `comp`, `sci`, `talk`
- 70% train / 30% test
- feature dim.: 700K (all 2-grams)



comp



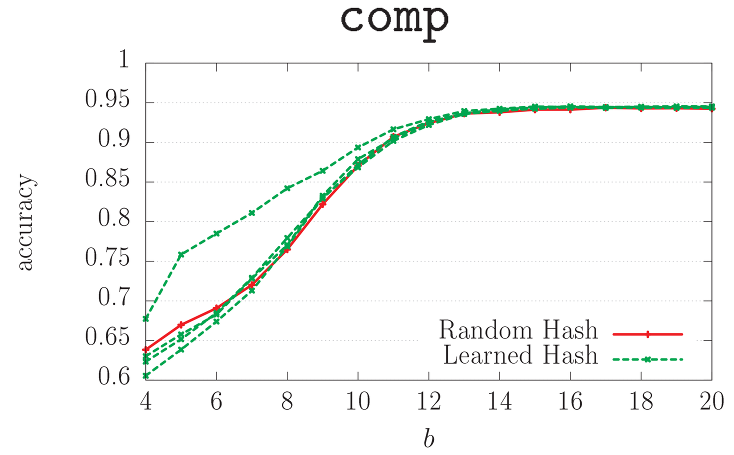
sci



talk

★ dataset #2: **RCV1**

- hashes learned on 100K examples
- Vowpal Wabbit ran on full train set
- feature dim.: 40K



- vectors $\boldsymbol{\nu}$ filled with 1/2-gram **DICE** coefficients: $\nu'_d(d) = \text{dice}(d, d')$ for all d, d'
- Vowpal Wabbit was trained on preprocessed data (with learned hashes)
- plots: classification **accuracy vs. # of bits** b
- **the higher the curve, the better**

Conclusion

For the considered **classification** task:

- the less RAM is available (the smaller b)
- or the higher dim. input feature space has

.. the more sense it makes to learn hashes

Future work:

- ranking objectives
- tasks with very large dimension:
  - information retrieval
  - collaborative filtering

References

[Ganchev and Dredze, 2008] Ganchev, K. and Dredze, M. (2008). Small statistical models by random feature mixing. In *Proceedings of the ACL-2008 Workshop on Mobile Language Processing*. Association for Computational Linguistics.

[Kushilevitz et al., 1998] Kushilevitz, E., Ostrovsky, R., and Rabani, Y. (1998). Efficient search for approximate nearest neighbor in high dimensional spaces. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, STOC '98, pages 614–623, New York, NY, USA. ACM.

[Shi et al., 2009] Shi, Q., Petterson, J., Dror, G., Langford, J., Smola, A. J., and Strehl, A. L. (2009). Hash Kernels. 5:496–503.

[Weinberger et al., 2009] Weinberger, K., Dasgupta, A., Langford, J., Smola, A. J., and Attenberg, J. (2009). Feature hashing for large scale multitask learning. *Computing Research Repository*, abs/0902.2:140–1120.