Task-driven Greedy Learning of Feature Hashing Functions

Artem Sokolov **Stefan Riezler**
Department of Computational Linguistics
Heidelberg University
69120 Heidelberg, Germany
{sokolov,riezler}@cl.uni-heidelberg.de

Abstract

Randomly hashing multiple features into one aggregated feature is routinely used in large-scale machine learning tasks to both increase speed and decrease memory requirements, with little or no sacrifice in performance. In this paper we investigate whether using a learned (instead of a random) hashing function improves performance. We show experimentally that with increasing difference between the dimensionalities of the input space and the hashed space, learning hashes is increasingly useful compared to random hashing.

1 Introduction

Standard approaches to learning from non-vectorial sparse data start by embedding data in a high-dimensional vector space $\mathbb{R}^D$ using domain-specific keys as dimension indexes: bag-of-tokens in Natural Language Processing (NLP), pairs of all possible query-document token associations in Information Retrieval (IR), all action-user pairings in collaborative filtering etc. For an embedded vector $\boldsymbol{\phi} \in \mathbb{R}^D$, and the widely used linear parametrization of the prediction (scoring) function $f(\boldsymbol{\phi}; \mathbf{w}) = \langle \mathbf{w}, \boldsymbol{\phi} \rangle$, the high number of dimensions often allows approximately achieving linear separability in classification tasks at an acceptable level. In practice however, the same high number of features can cause learning on sparse and non-integer indexed data to be both space and performance challenging. On the one hand, efficiently manipulating a model $\mathbf{w}$ during learning in most cases translates into the requirement to keep the complete model in quickly accessible memory. Depending on the hardware a learner must be run on, $\mathbf{w}$ of even a relatively modest dimension D may not easily fit into device's addressable RAM. On the other hand, naive storage of sparse data in ordered associative arrays or unordered hash tables, that bind non-integer feature keys to numerical values, becomes progressively slower with increasing dimensionality D.

The common approach to deal with both problems is the *hashing trick* (also called alphabet elimination or random feature mixing) [1, 2, 3]. The idea is to use a data-independent (pseudo-) random hashing function $HASH$ to define a per-coordinate mapping from $\mathbb{R}^D$ into a lower dimensional feature space $\mathbb{R}^M$:

$$\text{for } d' = 1 \ldots M \quad \phi'_{d'} = \sum_{d:HASH(d)=d'} \phi_d \qquad \text{or} \qquad \phi'_{d'} = \sum_{d:HASH(d)=d'} \xi_d \phi_d, \tag{1}$$

where d is a feature key (usually a string) and ξ_d are independent uniformly distributed binary (+1/-1) Bernoulli variables [3]. Both mappings provide, respectively, biased and unbiased estimators for inner products $\langle \boldsymbol{\phi}'_n, \boldsymbol{\phi}'_m \rangle$ in the reduced space [2, 3]; and both enjoy exponential tail bounds on the distortion of products [2, 3] or, in other words, lengths of vectors are approximately preserved with high probability.

In this work we design a *data-dependent* feature hashing technique to answer the question if better classification performance can be achieved with a mapping $HASH$ that is better informed of the final learning objective. The (ideal) solution that we are aiming for, should leverage existing highly optimized learning procedures: in the simplest case, it should be implementable as a preprocessing step that can be prepended early in the data processing pipeline and does not interfere with black-box online algorithms that optimize model $\mathbf{w}$.

We achieve this by requiring that for each feature d there exists a separate vector representation $\boldsymbol{\nu}(d) \in \mathbb{R}^V$. E.g., for word features, each word d could be accompanied with various NLP or IR statistics describing its usage in the wild, collected into $\boldsymbol{\nu}(d)$ over some text corpus not necessarily related to the task being solved. Based on the $\boldsymbol{\nu}$-representation we propose to, first, learn a mapping $\mathcal{H} : \mathbb{R}^V \to \{0,1\}^T$, such that the Hamming distance $D_H(\mathcal{H}(\boldsymbol{\nu}(d_1)), \mathcal{H}(\boldsymbol{\nu}(d_2)))$ between two features d_1, d_2 captures their similarity for the task. Second, a surjective mapping $\mathcal{B} : \{0,1\}^T \to \{0, \dots M\}$ is done, which has a property that sufficiently close (according to D_H) vectors get projected into the same integer. Finally, new hashing function $HASH$ is defined as the composition $\mathcal{H} \circ \mathcal{B}$ whose outputs can be interpreted as memory addresses.

Related work Oblivious (*data-independent*) locality-sensitive hashes [4] were initially proposed for the task of nearest-neighbor search and for specific (dis)similarity measures: Hamming distance [4, 5], ℓ_2 [5], ℓ_p [6], cosine [7], resemblance [8], edit distance [9] etc. Supervised *data-dependent* learning-to-hash approaches were later shown to significantly outperform LSH-based techniques (e.g., [10, 11]). Contrary to the oblivious approaches, here a measure to preserve is not known beforehand, rather it is inferred from examples of close/distant data instances obtained either from human labels or class memberships. A straight-forward application of the learning-to-hash to feature hashing would be to use domain-specific feature similarity, e.g. semantic closeness in NLP. Having clustered the feature vocabulary into classes, class id can be used as hash value in (1). In [12] correlated feature hashing using top-k words with highest DICE coefficient as similarity class fingerprint was indeed shown to be helpful, but in general this approach requires considerable trial-and-error effort. As the relation of the hash learning and the actual task (2)) is absent, one may miss some deeply hidden and not evident relations between words that would be advantageous to the task. Boosting was previously used to learn data-dependant binary hashes (e.g. in [10, 13]) with a different task to preserve pair-wise similarity and requiring a separate set of similarity-labelled data, which is absent in our setup.

2 Learning to Hash Features

Setup Suppose the training data is given in the form of N vectors $\boldsymbol{\phi}_n \in \mathbb{R}^D$ paired with a binary (± 1) class label y_n: $\{(\boldsymbol{\phi}_n, y_n)\}_{n=1\dots N}$. For NLP tasks, an individual component ϕ_d can be an indicator or frequency counts of the corresponding token d from a vocabulary of size D. For each feature d is accompanied by a representation $\boldsymbol{\nu}(d) \in \mathbb{R}^V$. These vectors provide an apriori view of the input feature vocabulary relatively to some, in general, independent feature set of size V. We consider a supervised task of finding optimal parameter vector $\mathbf{w}^* \in \mathbb{R}^M$ that minimizes training empirical loss

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} L(\mathbf{w}) = \arg\min_{\mathbf{w}} \sum_n \ell(\langle \mathbf{w}, \boldsymbol{\phi}'_n \rangle, y_n), \tag{2}$$

and a setup where the efficiency and/or resource constrains suggest using the hashing trick. Here $\ell(\langle \mathbf{w}, \boldsymbol{\phi}'_n \rangle, y_n)$ is the instance penalty suffered for predicting $\langle \boldsymbol{\phi}'_n, \mathbf{w} \rangle$ instead of y_n and the transformed vectors $\boldsymbol{\phi}'$ are obtained using one of the mappings (1). In the following we focus on the hinge loss $(1 - y\langle \mathbf{w}, \boldsymbol{\phi}' \rangle)_+$ and, for brevity, omit a regularization term that is likely to be needed in practice.

MQP formulation The unknown hashing function $HASH$ can be represented by a binary matrix $A \in \{0,1\}^{M \times D}$, where $a_{m,d} = [\![HASH(d) = m]\!]$. By definition A is sparse and has a special structure: for each d'th input feature, the d'th column of A contains exactly one 1; all other cells in the column are zero. As normally $M \ll D$, some rows will contain several 1's. The task of learning a data-dependant function $HASH$ (matrix A) and $\mathbf{w}^*$ simultaneously can be formulated as a mixed quadratic programming (MQP) problem:

$$\min_{\mathbf{w}, A, \boldsymbol{\xi}} \sum_n \xi_n, \text{ s.t.} \quad y_n \langle A^\top \mathbf{w}, \boldsymbol{\phi}_n \rangle \geq 1 - \xi_n, \quad \xi_n \geq 0, \quad \sum_m a_{m,d} = 1, \quad a_{m,d} \in \{0,1\} \qquad \forall n, m, d \tag{3}$$

Because of the size of the problem, outsourcing the task to QP solvers is hard: e.g., for cross-lingual information retrieval [12, 14] or collaborative filtering [15] D is easily in the billions and M in tens/hundreds of millions. If $\mathbf{w}$ is fixed (e.g., found separately for fixed A) we still obtain a hard Boolean LP task over $a_{m,d}$ that are known to be NP-complete in general [16] and the size of the problem is not handled by available ILP solvers.

The complexity of the MQP formulation (3) stems, on the one hand, from the over-generality of A that permits any mappings of even related features into arbitrary hash values, something that runs contrary to the idea of a useful data-dependant hashing. On the other hand, directly considering $a_{m,d}$ as variables makes the optimization

task discrete and, therefore, non-smooth. We will use two distinct mappings, to separate the similarity-modeling part of A and its non-smooth part that actually produces the integer indexes. First, we restrict possible A by grounding them on a mapping $\mathcal{H}: \boldsymbol{\nu} \mapsto [h_1(\boldsymbol{\nu}), \dots, h_T(\boldsymbol{\nu})]$ from $\mathbb{R}^V$ into a proxy Hamming space $\{0,1\}^T$ that hopefully reflects task-similarity of different $\boldsymbol{\nu}$'s in the Hamming distance D_H. Hamming space is a convenient tool to approximate the similarity of features represented by $\boldsymbol{\nu}(d)$ by progressively increasing dimension T until satisfactory quality is reached, also it is well studied and has efficient dimensionality reduction techniques [5, 7, 4]. Second, we fix a subsequent embedding $\mathcal{B}: \{0,1\}^T \to [0 \dots M]$ to be a dimensionality reducing mapping such that $P[\mathcal{B}(\boldsymbol{h}_1) = \mathcal{B}(\boldsymbol{h}_2)]$ is high for $\boldsymbol{h}_1, \boldsymbol{h}_2$ such that $D_H(\boldsymbol{h}_1, \boldsymbol{h}_2)$ is small. The values of the last mapping are interpreted as memory addresses, so M is dictated by the available memory size. After the hash function $HASH$ is learned it is fixed and any standard learning algorithm can be applied to solve (2).

First mapping: greedy learning of Hamming vectors We use the following intuition: the smaller is $|w_d - w_{d'}|$ for two distinct features d, d', the less loss distortion is incurred by hashing them in (1) into the same value $HASH(d) = HASH(d')$. In the extreme case $w_d = w_{d'}$, the loss remains the same after modifying $HASH$ to make the two features collide. To reflect proximity of w_d with adjustable accuracy we use Hamming vectors $\mathcal{H}(\boldsymbol{\nu}(d)) = [h_1(\boldsymbol{\nu}(d)), \dots, h_T(\boldsymbol{\nu}(d))]$ for each d and look for weights representable as $w_d^T = \sum_{t \le T} \alpha_t h_t(\boldsymbol{\nu}(d))$. If we now set all $\alpha_t = \alpha$, then $D_H(\mathcal{H}(\boldsymbol{\nu}(d)), \mathcal{H}(\boldsymbol{\nu}(d'))) = (w_d - w_{d'})/\alpha$. Hence, close $\mathcal{H}(\boldsymbol{\nu}(d))$ for different d would correspond to close[1] values of respective w_d. Empirical loss rewrites as:

$$L(\{\alpha_t, h_t\}) = \sum_n (1 - y_n \sum_d w_d^T \phi_{n,d})_+ = \sum_n (1 - y_n \sum_d \sum_{t \le T} \alpha_t h_t(\boldsymbol{\nu}(d)) \phi_{n,d})_+. \tag{4}$$

Learning of $\mathcal{H}(\boldsymbol{\nu}_d)$ can be done greedily in a boosting fashion. At step $t = 0$ the initial representation $\mathcal{H}^0(\boldsymbol{\nu})$ is empty. Let the representation on step $(t-1)$ be $\mathcal{H}^{t-1}(\boldsymbol{\nu}) = [h_1(\boldsymbol{\nu}), \dots, h_{t-1}(\boldsymbol{\nu})] \in \{0,1\}^{t-1}$. The $\mathcal{H}^t(\boldsymbol{\nu})$ is obtained from $\mathcal{H}^{t-1}(\boldsymbol{\nu})$ by appending a simple bit-function h_t from a family of weak bit-functions, in our case, the standard per-coordinate decision stumps: $h(\boldsymbol{\nu}; k, \theta) = [\![\nu_k > \theta]\!]$. Selecting $h(\boldsymbol{\nu}; k, \theta)$ is done seeking for a coordinate k and θ that has the largest absolute value of partial gradient of (4):

$$k^*, \theta^* = \arg\max_{k,\theta} \Big| \frac{\partial L}{\partial \alpha_t}|_{\alpha_t = 0} \Big| = \arg\max_{k,\theta} \Big| \sum_{n: y_n \langle \mathbf{w}^t, \boldsymbol{\phi}_n \rangle < 1} y_n \sum_d \phi_{n,d} [\![\nu_{d,k} \ge \theta]\!] \Big|.$$

The optimization can be efficiently done using for each k a set $\cup_d \{\nu_{d,k}\}$ presorted in descending order and inverted index for features d that return training instances d occurs in.

Second mapping: distance-sensitive projection After we have replaced joint learning of $\mathbf{w}$ and A with greedy learning of $\mathcal{H}(\boldsymbol{\nu}) = [h_1(\boldsymbol{\nu}), \dots, h_T(\boldsymbol{\nu})]$, we compress these potentially long vectors into shorter codes that have a high collision probability for close $\mathcal{H}(\boldsymbol{\nu})$. To that end, we use the Kushilevitz-Ostrovky-Rabani random traces [5] as bit representations of hash indexes. For a bit-vector $\boldsymbol{h} = [h_1, \dots, h_T]$ and vectors $\boldsymbol{r}_m = [r_{m,1}, \dots, r_{m,T}]$ of Bernoulli (0/1) random variables with probability parameter p, the trace $\boldsymbol{t}$ is a bit-vector $\boldsymbol{t} = [t_1, \dots, t_M]$, where $t_m = \langle \boldsymbol{h}, \boldsymbol{r}_m \rangle \bmod 2$. The probability for two vectors $\boldsymbol{h}_1$ and $\boldsymbol{h}_2$ to have identical traces decays monotonically with $D_H(\boldsymbol{h}_1, \boldsymbol{h}_2)$ and M:

$$P\big[\boldsymbol{t}_1 = \boldsymbol{t}_2 | D_H(\boldsymbol{h}_1, \boldsymbol{h}_2) \le \Delta\big] \ge \big(\frac{1}{2} + \frac{1}{2}(1-2p)^{\Delta}\big)^M. \tag{5}$$

As shown in [5] a single test bit t_m has a small bias towards closer $\boldsymbol{h}$, which is amplified by repeating the test M times. Decreasing p increases tolerance, i.e., the probability that distant bit-vectors will be hashed to the same M-bit hash. The memory address (new feature index) d' then has a binary M-bit representation: $d' = \sum_{m=1}^M 2^{m-1} \big[\langle \boldsymbol{r}_m, \boldsymbol{h} \rangle \bmod 2\big]$.

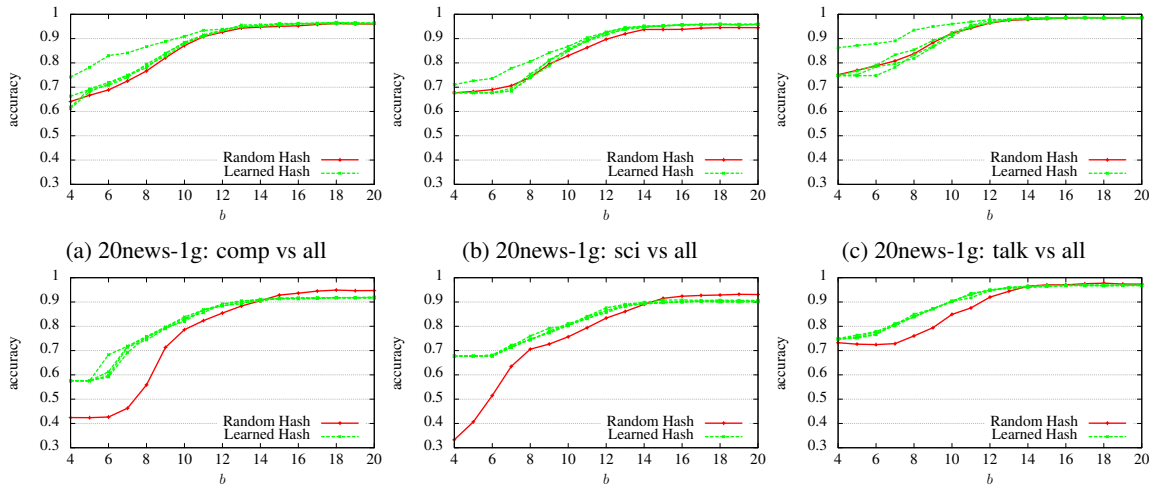
3 Experiments

For a proof-of-concept verification we used `20news` dataset[2] and the preprocessed version of the `RCV1-V2` dataset[3]. As baseline learning algorithm with a pseudo-random hashing function, we used the implementation

[1] We assume that α is a sufficiently small to accurately represent w.

[2] `http://qwone.com/~jason/20Newsgroups`

[3] `http://hunch.net/~vw/rcv1.tar.gz`



(a) 20news-1g: comp vs all (b) 20news-1g: sci vs all (c) 20news-1g: talk vs all

(d) 20news-2g: comp vs all (e) 20news-2g: sci vs all (f) 20news-2g: talk vs all

Figure 1: Test error rate for 3 classification tasks on `20news`. Different fusing curves for the learned hash correspond to different p in (5). The feature space is an order of magnitude larger in the experiments in the lower row, showing an increased gain for learned hashes, compared to smaller scale experiments in the upper row.

of hinge loss optimization in Vowpal Wabbit (VW) with 5 epochs, variable number of bits $b = 4, 5 \ldots 20$ for the hash length and default other settings. The same configuration was used to evaluate learned feature hash functions by first preprocessing training data with (1) and switching feature hashing off. For our approach we additionally equipped each (word or bigram) feature d with an array of, respectively, uni- or bi-gram DICE coefficients: $\nu_{d'}(d) = \text{dice}(d, d')$, collected over the training set. Using the learned models, the train and test data were preprocessed for $b = 4, 5 \ldots 20$ and $p = 0.5\%, 1\%, 5\%, 10\%$ to produce integer-indexed features.

20news Following [1, 17] we created 3 binary (one vs. all) tasks to classify between *comp*, *sci* and *talk* classes of the newsgroup hierarchy and randomly split them into training (70%, $N = 5,083$) and test (30%, $N = 2,178$) subsets. For each task two sparse ϕ representations were created: 1) bag-of-words weighted with tfidf and 2) binary bag-of-bigrams vectors, obtaining vocabulary sizes of $D_1 = 74K$ and $D_2 = 700K$, respectively. The boosting algorithm was run until $T_1 = 2,500$ and $T_2 = 3,500$ weak bits were accumulated for the uni-gram and bi-gram version, that contained on average, respectively, 278 and 282 unique weak learner per newsgroup type. Figure 1 shows the error rates for each of the task as a function of b.

RCV1 Experiments on the binary classification `RCV1-V2` dataset followed the same setup. Hash learning was done on the first $N = 100$K examples, obtaining a vocabulary of $D_3 = 40K$ elements, until $T_3 = 517$ (77 unique) weak hashes were obtained. For training VW we used the whole dataset (780K instances). Results are shown in Figure 2.

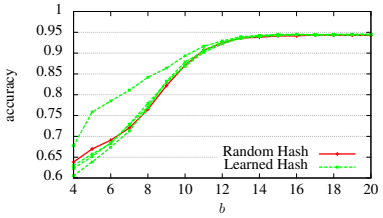


Figure 2: Test error rate for the `RCV1` classification task. Different fusing curves for the learned hash correspond to different p in (5). The upmost curve for the learned hash corresponds to $p = 0.005$.

4 Conclusion

We presented a boosting-style technique to learn hashing functions while optimizing a task-based objective. We showed experimentally that learning hashes is beneficial in the case when dimensionalities of the input and the hashed feature spaces differ substantially, and that the gains over random hashes increase with this difference. This can be explained by a higher damage from collisions of random hash functions that harms performance. Learning hash functions during task-driven optimization counterbalances the effects of more expressive feature spaces and collisions.

Acknowledgments

This work was supported in part by DFG grant "Cross-language Learning-to-Rank for Patent Retrieval".

References

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

[2] Qinfeng Shi, James Petterson, Gideon Dror, John Langford, Alexander J. Smola, and Alexander L. Strehl. Hash Kernels. 5:496–503, 2009.

[3] Kilian Weinberger, Anirban Dasgupta, John Langford, Alexander J. Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. *Computing Research Repository*, abs/0902.2:140–1120, 2009.

[4] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, STOC '98, pages 604–613, New York, NY, USA, 1998. ACM.

[5] Eyal Kushilevitz, Rafail Ostrovsky, and Yuval Rabani. 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, 1998. ACM.

[6] Mayur Datar and Piotr Indyk. Locality-sensitive hashing scheme based on p-stable distributions. In *In SCG 04: Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262. ACM Press, 2004.

[7] Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thirty-fourth annual ACM symposium on Theory of computing*, STOC '02, pages 380–388, New York, NY, USA, 2002. ACM.

[8] Andrei Z. Broder. On the resemblance and containment of documents. In *Proceedings of Compression and Complexity of Sequences (SEQUENCES'97*, pages 2–12. IEEE Computer Society, 1997.

[9] Artem Sokolov. Vector representations for efficient comparison and search for similar strings. *Cybernetics and System Analysis*, 43(4):484–498, July 2007.

[10] Gregory Shakhnarovich, Paul Viola, and Trevor Darrell. Fast pose estimation with parameter-sensitive hashing. In *Proceedings of the Ninth IEEE International Conference on Computer Vision - Volume 2*, ICCV '03, pages 750–, Washington, DC, USA, 2003. IEEE Computer Society.

[11] Yair Weiss, Antonio Torralba, and Robert Fergus. Spectral hashing. In *NIPS'08*, pages 1753–1760, 2008.

[12] Bing Bai, Jason Weston, David Grangier, Ronan Collobert, Kunihiko Sadamasa, Yanjun Qi, Olivier Chapelle, and Kilian Q. Weinberger. Learning to rank with (a lot of) word features. *Inf. Retr.*, 13(3):291–314, 2010.

[13] Antonio Torralba, Robert Fergus, and Yair Weiss. Small codes and large image databases for recognition. In *Proceedings of Computer Vision and Pattern Recognition (CVPR'08)*, page 18. IEEE, 2008.

[14] Artem Sokolov, Laura Jehl, Felix Hieber, and Stefan Riezler. Boosting cross-language retrieval by learning bilingual phrase associations from relevance rankings. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1688–1699. Association for Computational Linguistics, 2013.

[15] J. Bennett and S. Lanning. The netflix prize. In *Proceedings of the KDD Cup Workshop 2007*, pages 3–6, New York, August 2007. ACM.

[16] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. 1979.

[17] Amit Goyal, Piyush Rai, and Hal Daume III. Multiple hash functions for learning. In *Proceedings of the NIPS'11 Workshop on Big Learning*, 2011.