Bloom Features

Ashok Cutkosky
Computer Science Department
Stanford University
Stanford CA, USA
ashokc@stanford.edu

Kwabena Boahen
Bioengineering Department
Stanford University
Stanford CA, USA
boahen@stanford.edu

***Abstract*—We introduce a method for function-fitting that achieves high accuracy with a low memory footprint. For d-dimensional data and any user-specified m, we define a feature map from d to m dimensional Euclidean space with memory footprint $O(m)$ that scales as follows: As m increases, the space of linear functions on our m-dimensional features approximates any MAX (or boolean OR) function on the d-dimensional inputs with expected error inversely proportional to m. Our method is the only one in existence with this scaling that can simultaneously run in $O(m)$ time, process real-value inputs, and approximate non-linear functions, properties respectively not achieved by random Fourier features, b-bit Minwise Hashing, and Vowpal Wabbit, three competing methods. We achieve all three properties by using hashing ($O(m)$ space) to implement a sparse-matrix multiply ($O(m)$ time) with addition replaced by MAX (non-linear approximation). As these techniques are inspired by the Bloom filter, we call the vectors produced by our mapping Bloom features. We demonstrate that the scaling pre-factors are reasonable by testing our method on simulated (Dirichlet distributions) and real (MNIST and webspam) datasets.**

***Keywords*-large-scale learning, learning theory**

I. Learning with Memory Constraints

The use of randomness in order to save some resource such as memory (or computation time) at the potential expense of accuracy is an established strategy throughout computer science. A primary application of these low-memory footprint methods is in *large-scale* settings in which one must process an extremely large amount of possibly high-dimensional data. In this situation the original dataset may not fit into memory and so one turns to methods that reduce the memory footprint of the dataset but also allow the user to run some type of analysis on the data.

In machine learning, these memory savings are achieved by methods that use random projections [1] to approximate some function $f : \mathbb{R}^d \to \mathbb{R}$ using a small number of parameters, but state of the art methods suffer from particular drawbacks. Random Fourier features [2] has poor asymptotic time complexity; Vowpal Wabbit (VW) [3] can only approximate linear functions; and b-bit Minwise hashing [4] can only operate on binary inputs.

We present an algorithm that achieves a low memory footprint while avoiding these drawbacks. Similar to the previous methods, for any m we produce a mapping $\phi : \mathbb{R}^d \to \mathbb{R}^m$ such that as m increases linear functions on $\mathbb{R}^m$ approximate nonlinear functions in $\mathbb{R}^d$ with error inversely proportional to m. However our memory footprint increases linearly with m, and so m represents a trade-off between memory and accuracy in function fitting.

We call the vectors in $\mathbb{R}^m$ produced by our mapping Bloom features (Definition A.1), as our method was inspired by the Bloom filter data structure [5]. To compute Bloom features, we choose k hash functions to assign each component of a vector x in $\mathbb{R}^d$ to k components of $\phi(x)$ in $\mathbb{R}^m$. We compute the ith component of $\phi(x)$ by taking the MAX of all the components of x that are mapped to this ith component (Figure 1). This is analogous to multiplication by a sparse $m \times d$ matrix (specified by the k hash functions) with addition replaced by MAX. Using hash functions and sparsity allows us to have low memory footprint and small time complexity while the substitution of MAX for addition allows us to compute non-linear functions on real-valued inputs.

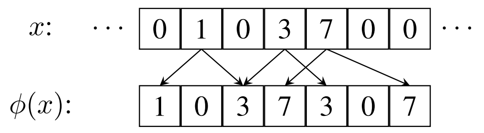

Fig. 1. Each coordinate of x is mapped to k random coordinates in $\phi(x)$ by hash functions. The final value of each coordinate of $\phi(x)$ is given by taking the maximum of the values hashed to it. In this case $k = 2$ and $m = 7$.

In Section II, we introduce Bloom features and describe their properties. In Section III, we empirically validate Bloom features on the freely available `MNIST` [6], and `webspam` [7] datasets. We also provide results on simulated data. We state formal results and proofs in an appendix.

II. Bloom Features

Approximating functions with Bloom features offers a good tradeoff between space, time and accuracy. Given input/output pairs $(x_i, y_i) \in \mathbb{R}^d \times \mathbb{R}$, this problem may be formulated as follows: Find $f_w : \mathbb{R}^d \to \mathbb{R}$ of the form $f_w(x) = \langle \phi(x), w \rangle$ where $\langle a, b \rangle$ indicates inner-product, such that $f_w(x_i) \approx y_i$, where $\phi : \mathbb{R}^d \to \mathbb{R}^m$ is a fixed map and $w \in \mathbb{R}^m$ is chosen to achieve the best approximation. The degree of approximation is measured by some loss function L. For example, $L(f_w, x, y) = (f_w(x) - y)^2$ is a common choice. Linear regression can be formulated in this manner with $\phi(x) = x$ and quadratic-kernel support vector regression corresponds to $\phi(x)$ equal to a vector of all 2nd order interactions in x.

Formulating function-approximation as a linear-map on a set of m-dimensional features $\phi(x)$, provided by a fixed mapping ϕ, introduces a tradeoff between resources and accuracy. As m increases, ϕ maps into higher dimensional spaces, resulting in more accurate function-fitting through inner products with $\phi(x)$, but requiring more time and space to be computed. A desirable ϕ will obtain a good tradeoff between space, time and accuracy.

The Bloom feature map achieves a good tradeoff by using hash functions to specify a sparse matrix and replacing the addition in matrix multiplication by MAX. This map ϕ operates on d-dimensional non-negative inputs—that is, elements of $\mathbb{R}^d_{\geq 0}$. It can approximate any linear combination of binary OR functions or real-valued MAX functions with error that decreases inversely proportional to m (Theorems A.4, A.8) and can be computed in $O(m)$ space and $O(m)$ time (Proposition A.2). While its space scaling is the same as three state-of-the-art methods, its time scaling is n to n/m times better (Table I).

TABLE I
Comparison between Methods

Feature	Time	Non-Binary	Non-linear
Bloom	$O(m)$	Yes	Yes
Fourier	$O(nm)$	Yes	Yes
b-bit Minwise	$O(\frac{nm}{2^b})$	No	Yes
VW	$O(n)$	Yes	No

III. Empirical Results

To verify that Bloom features' resource scaling factors are reasonable and confirm that its theoretical performance generalizes to the real-world, we tested it on simulated data (binary and real-valued) and real data (`MNIST` and `webspam`).

A. Simulated Data

We first tested Bloom features on a simulated classification task with binary vectors. The binary vectors were drawn from $\{0, 1\}^{100}$ with 10 randomly selected non-zero entries. The two classes were defined by an OR function computed on either 3, 5 or 10 randomly chosen bits. We chose a subset of the data that had an equal fraction belonging to each class. We used Bloom features with $m = 100$, 1000, 10000 and $k = \lfloor \frac{m}{10} \log(2) \rfloor$, the optimal value for a Bloom filter. We found that Bloom features achieve low MSE using m values significantly smaller than $\binom{100}{F}$, the number of possible OR functions of fan-in F (Figure 2).

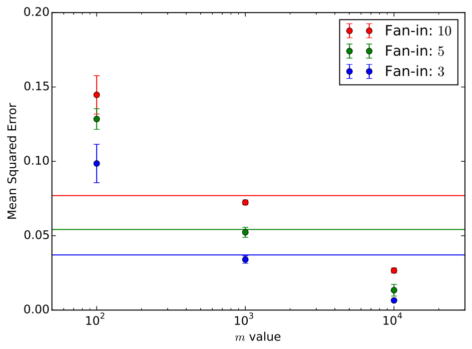

Fig. 2. Mean squared error vs m for a boolean OR of fan-in 3 and 10, with $d = 100$. Horizontal lines indicate best linear regression error for comparison. Error bars represent one standard deviation.

Next, we tested Bloom features on a simulated real-valued classification task consisting of two classes defined by two distinct, 100-dimensional Dirichlet distributions with parameters α and $\frac{3\alpha+\beta}{4}$ respectively. α and β were drawn uniformly at random from $(0, 1)^{100}$. This particular choice results in a Bayes risk of 0.3% (error of optimal classifier). We compared Bloom features to random Fourier features with m ranging from 50 to 10000 for both. k was set to $\lfloor \frac{m}{10} \log(2) \rfloor$ for Bloom features; b-bit Minwise hashing was not included because this method only applies to binary inputs. We found that Fourier features were more accurate for small m whereas Bloom features were more accurate for large m (Figure 3).

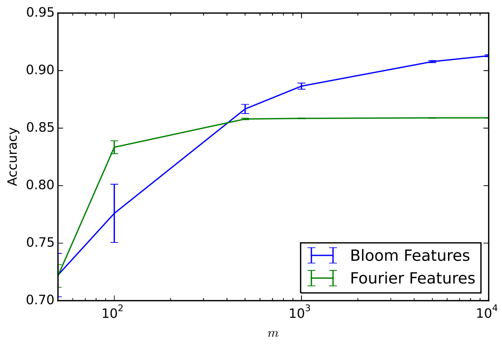

Fig. 3. Bloom features vs. random Fourier Features on simulated dataset. Error bars represent one standard deviation.

We then proceeded to test Bloom features on more realistic data by classifying the MNIST [6] and webspam [7] datasets. For all tasks, we applied one-vs-all linear classification on a Bloom feature representation of the data.

B. MNIST

The MNIST dataset contains 70000 28×28 gray-scale images of handwritten digits 0 through 9, which are split into a training and test set of size 60000 and 10000 respectively. The task is to determine which digit a given image represents. We consider the permutation-invariant form of the task, in which one cannot take advantage of any previously known structure in the data (such as the fact that the inputs are 2D images). We chose k via cross-validation on $m = 400$, which gave $k = 4$. We therefore used $k = m/100$ (rounded to the nearest integer) for all other values of m.

We compared Bloom features' performance on MNIST with that of random Fourier Features and b-bit Minwise hashing, which have the same memory footprint. We did not include VW because it can only classify linearly separable data; it is well-known that MNIST is not linearly separable. In the case of Minwise hashing, we converted the images to binary vectors by thresholding values to 1, an easy and natural process for MNIST. For other datasets (e.g. our simulation in Figure 3), this may not be true. We found that the performance gap between Bloom and Fourier features narrowed as m increased (Table II), as expected from our results with simulated data (Figure 3). And both out-performed Minwise hashing, demonstrating the advantage of using real-valued vectors.

We also compared Bloom features' performance on MNIST with several state-of-the-art algorithms that have a much larger memory footprint as measured by parameter count (taken from Table 1 of [8]). One of these algorithms (Maxout MLP, described in [8]) is similar in spirit to Bloom features. In this method, one trains a multi-layer perceptron whose activation function takes the maximum of its inputs. We found that Bloom features use 95% fewer parameters than any of these algorithms while achieving an error-rate within a factor of two of the lowest result (1.6% versus 0.79%).

TABLE II
MNIST PERFORMANCE (AVERAGE ERROR)

m	Bloom	Fourier	b-bit Minwise
400	6.3%	5.3%	11.1%
1000	3.8%	3.1%	7.4%
10000	1.8%	1.6%	2.8%

TABLE III
MNIST PERFORMANCE (OTHER METHODS)

Method	Error	Parameter Count
ReLU MLP [9]	1.05%	1794000
Maxout MLP [8]	0.94%	2392800
Manifold Tangent Classifier [10]	0.81%	5588000
DBM (with dropout) [11]	0.79%	2392800
Bloom Features	1.6%	100000

C. webspam

The webspam [7] dataset consists of 350000 sparse vectors of trigram counts representing either spam or non-spam documents. The data have a dimensionality of 16609143, of which only 680715 components are ever nonzero. webspam is nearly linearly separable, with linear classification accuracies in excess of 99%. Thus, we consider classifying webspam as a demonstration that Bloom features are still able to capture linear relationships. To classify webspam we selected 80% of the data to be a training set and 20% to be a testing set.

We compared Bloom features' performance on webspam with that of random Fourier Features, b-bit Minwise hashing, and VW for $m = 100, 1000, 10000$. All three choices use much less memory than is needed to represent the original features (up to 10000 vs 680715). `webspam`'s sparse binary features and linearly separability are exactly the conditions required by b-bit Minwise hashing and VW, respectively, hence they performed the best (Table IV). Nevertheless, across the range of m values tested, Bloom Features' error was no worse than 1.625 times that of these methods.

TABLE IV
WEBSPAM PERFORMANCE (AVERAGE ERROR)

m	Bloom	Fourier	b-bit Minwise	VW
100	9.4%	26.4%	10.8%	8.1%
1000	2.9%	14.7%	2.1%	1.9%
10000	1.3%	7.9%	0.8%	1.1%

IV. CONCLUSIONS

We have introduced Bloom features and analyzed their performance. Bloom feature representations are memory-efficient ($O(m)$) and can be computed very quickly ($O(m)$). As their dimensionality m increases, they approximate boolean functions and non-linear real-valued functions with error decreasing inversely proportionally to m. We prove these results in the Appendix and show that Bloom features operate by approximating a high-dimensional RKHS consisting of all interactions among inputs.

We demonstrated the practical viability of Bloom features using simulated data and the `MNIST` and `webspam` datasets. On the simulated and `MNIST` datasets, Bloom features represented non-linear functions on real-valued inputs accurately and efficiently. On the `webspam` dataset, Bloom features were competitive with more restrictive methods such as b-bit Minwise hashing and VW when representing linear functions on binary inputs. Thus Bloom features not only possess theoretical advantages in terms of either handling real-valued inputs, computation time, or non-linear approximation over other methods, they also compare favorably on practical tasks in terms of test error for a given memory footprint.

REFERENCES

[1] William B Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26(189-206):1, 1984.

[2] Ali Rahimi and Benjamin Recht. Random Features for Large Scale Kernel Machines. In *Advances in Neural Information Processing Systems*, 2007.

[3] W Kilian, D Anirban, L John, S Alex, and A Josh. Feature Hashing for Large Scale Multitask Learning Feature Hashing for Large Scale Multitask Learning. In *International Conference on Machine Learning (ICML)*, pages 1113–1120, 2009.

[4] Ping Li and Christian König. b-bit minwise hashing. In *Proceedings of the 19th international conference on World wide web*, pages 671–680. ACM, 2010.

[5] B Bloom. Space/Time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[6] L Yann, B Léon, B Yoshua, and H Patrick. Gradient-Based Learning Applied to Documnet Recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[7] Steve Webb, James Caverlee, and Calton Pu. Introducing the webb spam corpus: Using email spam to identify web spam automatically. In *CEAS*, 2006.

[8] G I J, W David, M Mehdi, C Aaron, and B Yoshua. Maxout Networks. *Journal of Machine Learning Research (JMLR)*, 28(3):1319–1327, 2013.

[9] Nitish Srivastava. *Improving neural networks with dropout*. PhD thesis, University of Toronto, 2013.

[10] Salah Rifai, Yann N Dauphin, Pascal Vincent, Yoshua Bengio, and Xavier Muller. The manifold tangent classifier. In *Advances in Neural Information Processing Systems*, pages 2294–2302, 2011.

[11] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[12] Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.

APPENDIX

Definition A.1. Bloom Features *Suppose* $h_1, \dots, h_k$ *are functions mapping* $\mathbb{N}$ *to* $\{0, \dots, m-1\}$ *drawn at random from a pair-wise independent family of hash functions. For* $x \in \mathbb{R}^d_{\geq 0}$*, define* $\phi_h(x) \in \mathbb{R}^m_{\geq 0}$ *by*

$$\phi_h(x)_i = \max_{j,l:\ h_l(j)=i} x_j$$

where the subscript h *in* ϕ_h *is intended to emphasize that* $\phi_h(x)$ *depends on the choice of hash functions.*

Proposition A.2. Computation *Suppose* $x \in \mathbb{R}^d$ *with* n *non-zero components, and let* $\phi_h(x)$ *be a Bloom feature map using* k *hash functions. Then* $\phi_h(x)$ *can be computed in* $O(kn)$ *time when vectors are represented in sparse format (only non-zero quantities stored), and in* $O(d + kn + m)$ *time when vectors are represented in dense format.*

Proposition A.3. Memory *Standard Bloom filter analysis [12] suggests a value for k: If our inputs* x *have* n *non-zero components, then we should set* $k = \log(2)\frac{m}{n}$*. Using this value of* k *and Proposition A.2, we get a complexity of* $O(kn) = O(m)$*.*

Theorem A.4. Binary Input Approximation *Suppose* $B(x)$ *is a boolean function of fan-in* F *and*

$\phi_h(x) : \mathbb{R}^d \to \mathbb{R}^m$ is a Bloom feature map with k hash functions (see Definition A.1). Let n be the number of set bits of some vector $x \in \{0,1\}^d$. Let $p = 1 - (1 - \frac{1}{m})^k$ be the probability that a bit of x is hashed to a given component of $\phi_h(x)$. Then for sufficiently large m there exists $w_h \in \mathbb{R}^m$ such that

$$\mathop{\mathbb{E}}_h[\phi_h(x) \cdot w_h] = B(x)$$
$$\mathop{\mathbb{E}}_h[(\phi_h(x) \cdot w_h - B(x))^2] = O\left(\tfrac{2^F}{p^F(1-1/m)^{kn}m}\right)$$

Proof: Our proof has two steps. First we observe that we can write $B(x)$ as a linear combination of ORs involving no negations: $B(x) = \sum_i^s a_i(x_{i_1} \vee \cdots \vee x_{i_z})$ with $a_i \in \mathbb{R}$, $z \leq F$ and $s \leq 2^F$. Then we show that $x_{i_1} \vee \cdots \vee x_{i_z} \approx \phi_h(x) \cdot w_{h,i}$ for some appropriate $w_{h,i}$ (see Lemma A.5). Substituting this approximation in to the expression for $B(x)$ completes the proof with $w_h = \sum_i a_i w_{h,i}$.

To prove the first step, we consider OR functions as vectors in $\mathbb{R}^{2^F}$ (defined by their truth tables), and show that they are linearly independent in this space. To see this, first set $y_j = 1 - x_j$. Then by De Morgan's rules we have $x_{i_1} \vee \cdots \vee x_{i_z} = 1 - y_{i_1} \cdots y_{i_z}$. Thus there is a 1-1 linear map from OR functions to monomials $y_{i_1} \cdots y_{i_z}$. Now since the set of distinct monomials is linearly independent, so are the OR functions. ■

Lemma A.5. *Suppose $C^1(x)$ and $C^2(x)$ are OR functions of fan-in F^1 and F^2. Further, define $F^\cup$ as the number of inputs shared by C^1 and C^2 and let $\overline{F} = (F^1 + F^2) - F^\cup$. Suppose $\phi_h(x)$ is a Bloom feature map with k hash functions. Then for sufficiently large m there exists w_h^1 and w_h^2 such that:*

$$\mathop{\mathbb{E}}_h[\phi_h(x) \cdot w_h^i] = C^i(x)$$
$$\mathit{Var}(\phi_h(x) \cdot w_h^i) \leq \tfrac{4}{m} p^{-F^i}(1-p)^{-n}$$
$$\mathit{Cov}(\phi_h(x) \cdot w_h^1, \phi_h(x) \cdot w_h^2) \leq \tfrac{2}{m} p^{-\overline{F}}(1-p)^{-n}$$

where p, m, n, x were defined previously (see Theorem A.4).

Proof: If $\phi_h(x)$'s components were both zero-mean and independent, then we can prove this result with bounds improved by a factor of 4 (Lemma A.7). To prove this result for the non-zero-mean, non-independent ϕ_h, we subtract $\phi_h(x)$'s first $m/2$ components from its second $m/2$ components to form a new vector $\phi_h(x)'$ that is half the size but is now zero-mean while still having non-independent components. Since $\phi_h(x)'$ is half the size, we lose a factor of 2 in the bounds. Now we complete the proof by showing that the lack of independence between components introduces another factor of 2 in the bounds.

The factor of 2 loosening due to the lack of independence comes from distributing expectations over products. Specifically, we need to show that distributing the expectation $\mathbb{E}_h[\phi_h(x)_i \phi_h(x)_j]$ over the product accrues a error that goes down as m increases. To do this we show that $\phi_h(x)_i$ concentrates about its mean. Let e be the number of zero bits in $\phi_h(x)$ and $q = \frac{e}{m}$. Mitzenmacher and Upfal show [12] for Bloom filters (which applies to Bloom features in the binary case) that $\mathbb{P}(|q - \mathbb{E}[q]| > \frac{\lambda}{m}) < 2\exp(-2\lambda^2/m)$. Thus

$$\mathbb{E}[q^2] \geq (1 - 2\exp(-2\lambda^2/m))(\mathbb{E}[q] - \lambda/m)^2$$
$$\mathbb{E}[q^2] \leq (1 - 2\exp(-2\lambda^2/m))(\mathbb{E}[q] + \lambda/m)^2 + 2\exp(-2\lambda^2/m)$$

So that for any δ, for sufficiently large m, we must have $\frac{1}{1-\delta}\mathbb{E}[q]^2 \leq \mathbb{E}[q^2] \leq (1+\delta)\mathbb{E}[q]^2$. Now

$$\mathbb{E}[\phi_h(x)_i \phi_h(x)_j] = \sum_t \mathbb{P}(q = t)(1-t)(1 - t\tfrac{m}{m-1})$$
$$= 1 - \tfrac{2m-1}{m-1}\mathbb{E}[q] + \tfrac{m}{m-1}\mathbb{E}[q^2]$$

Thus for any δ', for sufficiently large m, $\mathbb{E}[\phi_h(x)_i \phi_h(x)_j]$ is within a factor of $1 + \delta'$ of $(1 - \mathbb{E}[q])^2 = \mathbb{E}[\phi_h(x)_i]\,\mathbb{E}[\phi_h(x)_j]$. Choosing $\delta' = 1$, for sufficiently large m the error from the independence assumption is bounded by a factor of 2. ■

Definition A.6. Independent Features *Let $p \in (0,1)$. Suppose $z^+_{i,j}, z^-_{i,j}$ are independent Bernoulli random variables with the same mean p for $i \in \{0, \ldots, m-1\}$, $j \in \{0, \ldots, d-1\}$. Then define*

$$T^+(x)_i = \max_{j : z^+_{i,j} = 1} x_j$$
$$T^-(x)_i = \max_{j : z^-_{i,j} = 1} x_j$$
$$\phi_z(x)_i = T^+(x)_i - T^-(x)_i$$

where here the subscript z indicates that ϕ_z depends on the values of the variables $z^+_{i,j}$ and $z^-_{i,j}$.

Lemma A.7. *Let C^1, C^2, F^1, F^2, $F^\cup$, $\overline{F}$, x and n be as defined previously (Theorem A.4). Let $\phi_z(x)$ be an* *independent feature with parameter p (see Definition A.6). Then there exists $w_z^1, w_z^2 \in \mathbb{R}^m$ such that:*

$$\mathop{\mathbb{E}}_z[\phi_z(x)\cdot w_z^i] = C^i(x)$$
$$\mathit{Var}(\phi_z(x)\cdot w_z^i) \le \tfrac{1}{m}p^{-F^i}(1-p)^{-n}$$
$$\mathit{Cov}(\phi_z(x)\cdot w_z^1, \phi_z(x)\cdot w_z^2) \le \tfrac{1}{2m}p^{-\overline{F}}(1-p)^{-n}$$

Proof: We'll prove the bias and variance results for $C = C^1$. The results for C^2 are symmetric. Let $C = x_{c_1} \vee \cdots \vee x_{c_F}$.

We start by defining $Q_n = 1-(1-p)^n$, $Z_j^+ = \prod_{i=1}^F z_{c_i,j}^+$, and $Z_j^- = \prod_{i=1}^F z_{c_i,j}^-$. The following facts will be useful:

$$\mathbb{E}[Z_j^+] = \mathbb{E}[Z_j^-] = p^F \quad (1)$$
$$\mathbb{E}[Z_j^+ T^+(x)_j] = p^F C(x) + p^F(1-C(x))Q_n \quad (2)$$
$$\mathbb{E}[Z_j^+ T^-(x)_j] = p^F Q_n \quad (3)$$

With $(r_z)_j = Z_j^+ - Z_j^-$, some algebra gives:

$$\begin{aligned}\mathop{\mathbb{E}}_z[\phi_z(x)\cdot r_z] &= m\mathop{\mathbb{E}}_z[\phi_z(x)_1(r_z)_1] \\ &= 2m\mathop{\mathbb{E}}_z[T_1^+Z_1^+] - 2m\mathop{\mathbb{E}}_z[T_1^-Z_1^+] \\ &= 2mp^F(1-Q_n)C(x)\end{aligned}$$

For the variance, we compute

$$\mathrm{Var}(\phi_z(x)\cdot r_z) = m\mathrm{Var}(\phi_z(x)_1(r_z)_1)$$
$$\mathbb{E}[(\phi_z(x)_1(r_z)_1)^2] = \mathbb{E}[(Z^+-Z^-)^2(T_1^+-T_1^-)^2]$$

from which we obtain (from equations 1, 2, 3):

$$\mathrm{Var}(\phi_z(x)\cdot r_z) \le 4mp^F(1-Q_n)$$

Now if we set $(w_z)_j = \frac{1}{2}p^{-F}(1-Q_n)^{-1}\frac{1}{m}(r_z)_j$ then we recover the expectation and variance statements.

The covariance statement is computed similarly. ■

Theorem A.8. Approximating a space of MAX functions *Suppose $\phi_h(x)$ is a Bloom feature constructed with k hash functions. Suppose $K(x,y)$ is a scaled inner product of MAX functions as defined below. Then:*

$$\mathop{\mathbb{E}}_h[\tfrac{1}{\sqrt{m}}\phi_h(x)\cdot\tfrac{1}{\sqrt{m}}\phi_h(y)] = K(x,y)$$
$$\mathbb{P}\left(\sup_{x,y\in D_n}\left|\tfrac{\phi_h(x)}{\sqrt{m}}\cdot\tfrac{\phi_h(y)}{\sqrt{m}} - K(x,y)\right| > \epsilon\right) < \delta$$

as long as

$$m > \tfrac{2}{\epsilon^2}\left(\log(1/\delta) + 2\log\left(2\tbinom{d}{n}\right)\right)$$

Where $D_n \subset [0,1]^d$ is the set of vectors with at most n non-zero components.

$K(x,y)$ is given by the following construction. For $x \in [0,1]^d$, let $MAX_q(x) \in \mathbb{R}^{\binom{d}{q}}$ be the vector obtained by applying all q-ary MAX functions to the entries of x. Define $K(x,y) = \psi(x)\cdot\psi(y)$ with

$$\psi(x) = \bigoplus_{q=1}^{d}\sqrt{(1-p)^{d-q}p^q}\mathit{MAX}_q(x)$$

where $p = 1-(1-\frac{1}{m})^k$. Thus $K(x,y)$ is the kernel of the RKHS $\mathcal{H}$ defined by

$$\mathcal{H} \cong \bigoplus_{q=1}^{d}\mathbb{R}^{\binom{d}{q}} \cong \mathbb{R}^{2^d}$$

with $v(x) = v\cdot\psi(x)$ for $v \in \mathcal{H}$.

Proof: First we compute the expectation:

$$\mathop{\mathbb{E}}_h[\tfrac{1}{\sqrt{m}}\phi_h(x)\cdot\tfrac{1}{\sqrt{m}}\phi_h(y)] = \mathop{\mathbb{E}}_h[\phi_h(x)_1\phi_h(y)_1]$$

Let A_q be the event that there are exactly q values of t such that $h_l(t) = 1$ for some l. Then $P(A_q) = (1-p)^{d-q}p^q\binom{d}{q}$. Further,

$$\mathop{\mathbb{E}}_h[\phi_h(x)_1\phi_h(y)_1|A_q] = \tbinom{d}{q}^{-1}\mathrm{MAX}_q(x)\cdot\mathrm{MAX}_q(y)$$

$$\mathop{\mathbb{E}}_h\left[\tfrac{\phi_h(x)}{\sqrt{m}}\cdot\tfrac{\phi_h(y)}{\sqrt{m}}\right] =$$
$$\sum_{q=1}^{n}P(A_q)\tbinom{d}{q}^{-1}\mathrm{MAX}_q(x)\cdot\mathrm{MAX}_q(y) = K(x,y)$$

For the concentration inequality, we apply the Azuma-Hoeffding inequality to the Doob martingale given by

$$B_i = \frac{1}{m}\mathbb{E}[\phi_h(x)\cdot\phi_h(y)|\phi_h(x)_1,\ldots,\phi_h(y)_i]$$

Since each component of $\phi_h(x)$ is in $[0,1]$, $|B_{i+1}-B_i| \le 1$, and so

$$P\left(\left|\tfrac{\phi_h(x)}{\sqrt{m}}\cdot\tfrac{\phi_h(y)}{\sqrt{m}} - K(x,y)\right| > \epsilon\right) \le 2\exp(-\tfrac{\epsilon^2 m}{2})$$

The maximum error occurs at corners of D_n, so by applying a union bound over all $\binom{d}{n}^2$ pairs x,y we bound the probability of error more than ϵ by

$$2\tbinom{d}{n}^2\exp(-\epsilon^2 m/2) = \delta$$

Solve for m to prove the theorem. ■