Hash Kernels

Qinfeng Shi, James Petterson
Australian National University and NICTA,
Canberra, Australia

Gideon Dror
Department of Computer Science
Academic College of Tel-Aviv-Yaffo, Israel

John Langford, Alex Smola, Alex Strehl
Yahoo! Research
New York, NY and Santa Clara, CA, USA

Vishy Vishwanathan
Department of Statistics
Purdue University, IN, USA

Abstract

We propose hashing to facilitate efficient kernels. This generalizes previous work using sampling and we show a principled way to compute the kernel matrix for data streams and sparse feature spaces. Moreover, we give deviation bounds from the exact kernel matrix. This has applications to estimation on strings and graphs.

1 Introduction

In recent years, a number of methods have been proposed to deal with the fact that kernel methods have slow runtime performance if the number of kernel functions used in the expansion is large. We denote by $\mathcal{X}$ the domain of observations and we assume that $\mathcal{H}$ is a Reproducing Kernel Hilbert Space $\mathcal{H}$ with kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$.

Keeping the kernel expansion small One line of research (Burges and Schölkopf, 1997) aims to reduce the number of basis functions needed in the overall function expansion. This led to a number of reduced set Support Vector algorithms which work as follows: a) solve the full estimation problem resulting in a kernel expansion, b) use a subset of basis functions to approximate the exact solution, c) use the latter for estimation. While the approximation of the full function expansion is typically not very accurate, very good generalization performance is reported. The big problem in this approach is that the optimization of the reduced set of vectors is rather nontrivial.

Work on estimation on a budget (Dekel et al., 2006) tries to ensure that this problem does not arise in the first place by ensuring that the number of kernel functions used in the expansion never exceeds a given budget or by using an ℓ_1 penalty (Mangasarian, 1998). For some algorithms, e.g. binary classification, guarantees are available in the online setting.

Keeping the kernel simple A second line of research uses variants of sampling to achieve a similar goal. That is, one uses the feature map representation

$$k(x, x') = \langle \phi(x), \phi(x') \rangle . \tag{1}$$

Here ϕ maps $\mathcal{X}$ into some feature space $\mathcal{F}$. This expansion is approximated by a mapping $\bar{\phi} : \mathcal{X} \to \bar{\mathcal{F}}$

$$\bar{k}(x, x') = \left\langle \bar{\phi}(x), \bar{\phi}(x') \right\rangle \text{ often } \bar{\phi}(x) = C\phi(x), \tag{2}$$

where $C \in \mathbb{R}$. Here $\bar{\phi}$ has more desirable computational properties than ϕ. For instance, $\bar{\phi}$ is finite dimensional (Fine and Scheinberg, 2001; Kontorovich, 2007; Rahimi and Recht, 2008), or $\bar{\phi}$ is particularly sparse (Li et al., 2007).

Our Contribution Firstly, we show that the sampling schemes of Kontorovich (2007) and Rahimi and Recht (2008) can be applied to a considerably larger class of kernels than originally suggested — the authors only consider languages and radial basis functions respectively. Secondly, we propose a biased approximation $\bar{\phi}$ of ϕ which allows efficient computations even on data streams. Our work is inspired by the count-min sketch of Cormode and Muthukrishnan (2004), which uses hash functions as a computationally efficient means of randomization. This affords storage efficiency (we need not store random vectors) and at the same time they give performance guarantees comparable to those obtained by means of random projections.

As an application, we demonstrate computational benefits over suffix array string kernels in the case of document analysis and we discuss a kernel between graphs

Appearing in Proceedings of the 12^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2009, Clearwater Beach, Florida, USA. Volume 5 of JMLR: W&CP 5. Copyright 2009 by the authors.

which only becomes computationally feasible by means of compressed representation.

Outline We begin with a description of previous work in Section 2 and propose the hash kernels in Section 3. An analysis follows in Section 4 and we conclude with experiments in Section 5.

2 Previous Work and Applications

Generic Randomization Kontorovich (2007); Rahimi and Recht (2008) independently propose the following: denote by $c \in \mathcal{C}$ a random variable with measure P. Moreover, let $\phi_c : \mathcal{X} \to \mathbb{R}$ be functions indexed by $c \in \mathcal{C}$. For kernels of type

$$k(x, x') = \mathbf{E}_{c \sim \mathrm{P}(c)} \left[\phi_c(x)\phi_c(x')\right] \tag{3}$$

an approximation can be obtained by sampling $C = \{c_1, \ldots, c_n\} \sim \mathrm{P}$ and expanding

$$\bar{k}(x, x') = \frac{1}{n} \sum_{i=1}^{n} \phi_{c_i}(x)\phi_{c_i}(x'). \tag{4}$$

In other words, we approximate the feature map $\phi(x)$ by $\bar{\phi}(x) = n^{-1}\left(\phi_{c_1}(x), \ldots, \phi_{c_n}(x)\right)$. Assuming that $\phi_c(x)\phi_c(x')$ has bounded range, i.e. $\phi_c(x)\phi_c(x') \in [a, a + R]$ for all c, x and x' one may use Chernoff bounds to give guarantees for large deviations between $k(x, x')$ and $\bar{k}(x, x')$. For matrices of size $m \times m$ one obtains bounds of type $O(R^2 \epsilon^{-2} \log m)$ by combining Hoeffding's theorem with a union bound argument over the $O(m^2)$ different elements of the kernel matrix. The strategy has widespread applications beyond those of Kontorovich (2007); Rahimi and Recht (2008):

- Kontorovich (2007) uses it to design kernels on regular languages by sampling from the class of languages.
- The marginalized kernels of Tsuda et al. (2002) use a setting identical to (3) as the basis for comparisons between strings and graphs by defining a random walk as the feature extractor. Instead of exact computation we could do sampling.
- The Binet-Cauchy kernels of Vishwanathan et al. (2007) use this approach to compare trajectories of dynamical systems. Here c is the (discrete or continuous) time and $\mathrm{P}(c)$ discounts over future events.
- The empirical kernel map of Schölkopf (1997) uses $\mathcal{C} = \mathcal{X}$ and employs some kernel function κ to define $\phi_c(x) = \kappa(c, x)$. Moreover, $\mathrm{P}(c) = \mathrm{P}(x)$, i.e. placing sampling points c_i on training data.
- For RBF kernels (Rahimi and Recht, 2008) use the fact that k may be expressed in the system of eigenfunctions which commute with the translation operator, that is the Fourier basis

  $$k(x, x') = \mathbf{E}_{w \sim \mathrm{P}(w)}[e^{-i\langle w, x\rangle} e^{i\langle w, x'\rangle}]. \tag{5}$$

  Here $\mathrm{P}(w)$ is a nonnegative measure which exists for any RBF kernel by virtue of Bochner's theorem, hence (5) can be recast as a special case of (3). What sets it apart is the fact that the variance of the features $\phi_w(x) = e^{i\langle w, x\rangle}$ is relatively evenly spread. (5) extends immediately to Fourier transformations on other symmetry groups (Berg et al., 1984).
- The conditional independence kernel (Watkins, 2000) is one of the first instances of (3). Here $\mathcal{X}, \mathcal{C}$ are domains of biological sequences, $\phi_c(x) = \mathrm{P}(x|c)$ denotes the probability of observing x given the ancestor c, and $p(c)$ denotes a distribution over ancestors.

While in many cases straightforward sampling may suffice, it can prove disastrous whenever $\phi_c(x)$ has only a small number of significant terms. For instance, for the pair-HMM kernel most strings c are *unlikely* ancestors of x and x', hence $\mathrm{P}(x|c)$ and $\mathrm{P}(x'|c)$ will be negligible for most c. As a consequence the number of strings required to obtain a good estimate is prohibitively large — we need to reduce $\bar{\phi}$ further.

Locally Sensitive Hashing The basic idea of randomized projections (Indyk and Motawani, 1998) is that due to concentration of measures the inner product $\langle \phi(x), \phi(x') \rangle$ can be approximated by $\sum_{i=1}^{n} \langle v_i, \phi(x) \rangle \langle v_i, \phi(x') \rangle$ efficiently, provided that the distribution generating the vectors v_i satisfies basic regularity conditions. E.g. $v_i \sim \mathcal{N}(0, 1)$ is sufficient. This allows one to obtain Chernoff bounds and $O(\epsilon^{-2} \log d)$ rates of approximation. The main cost is to store v_i and perform the $O(nd)$ multiply-adds, thus rendering this approach too expensive as a preprocessing step in many applications.

Sparsification Li et al. (2007) propose to sparsify $\phi(x)$ by randomization while retaining the inner products. One problem with this approach is that when performing optimization for linear function classes, the weight vector w which is a linear combination of $\phi(x_i)$ remains large and dense, thus obliterating a significant part of the computational savings gained in sparsifying ϕ.

Count-Min Sketch Cormode and Muthukrishnan (2004) propose an ingenious method for representing data streams. Denote by $\mathcal{I}$ an index set. Moreover, let $h : \mathcal{I} \to \{1, \ldots, n\}$ be a hash function and assume that there exists a distribution over h such that they are pairwise independent.

Assume that we draw d hash functions h_i at random and let $S \in \mathbb{R}^{n \times d}$ be a sketch matrix. For a stream of symbols s update $S_{h_i(s),i} \leftarrow S_{h_i(s),i} + 1$ for all $1 \leq i \leq d$. To retrieve the (approximate) counts for symbol s' compute $\min_i S_{h_i(s'),i}$. Hence the name count-min sketch. The idea is that by storing counts of s according to several hash functions we can reduce the probability of collision with another particularly large symbol. Cormode and Muthukrishnan (2004) show that only $O(\epsilon^{-1} \log 1/\delta)$ storage is required for an ϵ-good approximation.

Cormode and Muthukrishnan (2004) discuss approximating inner products and the extension to signed rather than nonnegative counts. However, the bounds degrade for real-valued entries. Even worse, for the hashing to work, one needs to take the minimum over a set of inner product candidates. This breaks the dot product property.

Random Feature Mixing Ganchev and Dredze (2008) provide empirical evidence that using hashing can eliminate alphabet storage and reduce the number of parameters without severely impacting model performance. In addition, Langford et al. (2007) released the Vowpal Wabbit fast online learning software which uses a hash representation similar to that discussed here.

3 Hash Kernels

Our goal is to design a possibly biased approximation which a) approximately preserves the inner product, b) which is generally applicable, c) which can work on data streams, and d) which increases the density of the feature matrices (the latter matters for fast linear algebra on CPUs and graphic cards).

Kernel Approximation Our approach is most closely related to the Count-Min sketch (Cormode and Muthukrishnan, 2004) insofar as it uses a hash function to reduce a feature vector to a more compact representation. As before denote by $\mathcal{I}$ an index set and let $h : \mathcal{I} \to \{1, \ldots, n\}$ be a hash function drawn from a distribution of pairwise independent hash functions. Finally, assume that $\phi : \mathcal{X} \to \mathbb{R}^{|J|}$ is indexed by $\mathcal{I}$ and that we may compute $\phi_i(x)$ for all nonzero terms efficiently. In this case we define the hash kernel as follows:

$$\bar{k}(x, x') = \left\langle \bar{\phi}(x), \bar{\phi}(x') \right\rangle \text{ with } \bar{\phi}_j(x) = \sum_{i \in \mathcal{I}; h(i) = j} \phi_i(x) \quad (6)$$

We are accumulating all coordinates i of $\phi(x)$ for which $h(i)$ generates the same value j into coordinate $\bar{\phi}_j(x)$.

Our claim is that hashing preserves information as well as randomized projections with significantly less computation. Before providing an analysis let us discuss two key applications: efficient hashing of kernels on strings and cases where the number of classes is very high, such as categorization in an ontology.

Strings Denote by $\mathcal{X} = \mathcal{I}$ the domain of strings on some alphabet. Moreover, assume that $\phi_i(x) := \lambda_i \#_i(x)$ denotes the number of times the substring i occurs in x, weighted by some coefficient $\lambda_i \geq 0$. This allows us to compute a large family of kernels via

$$k(x, x') = \sum_{i \in \mathcal{I}} \lambda_i^2 \#_i(x) \#_i(x'). \quad (7)$$

Teo and Vishwanathan (2006) propose a storage efficient $O(|x| + |x'|)$ time algorithm for computing k for a *given* pair of strings x, x'. Here $|x|$ denotes the length of the string. Moreover, a weighted combination $\sum_i \alpha_i k(x_i, x)$ can be computed in $O(|x|)$ time after $O(\sum_i |x_i|)$ preprocessing.

The big drawback with string kernels using suffix arrays/trees is that they require large amounts of working memory. Approximately a factor of 50 additional storage is required for processing and analysis. Moreover, updates to a weighted combination are costly. This makes it virtually impossible to apply (7) to millions of documents. Even for modest document lengths this would require Terabytes of RAM.

Hashing allows us to reduce the dimensionality. Since for every document x only a relatively small number of terms $\#_i(x)$ will have nonzero values — at most $O(|x|^2)$ but in practice we will restrict ourselves to substrings of a bounded length l leading to a cost of $O(|x| \cdot l)$ — this can be done efficiently in a single pass over x. Moreover, we can compute $\bar{\phi}(x)$ as a preprocessing step and discard x altogether.

Note that this process spreads out the features available in a document *evenly* over the coordinates of $\bar{\phi}(x)$. Moreover, note that a similar procedure, or even the Count-Min sketch outright can be used to obtain good estimates for a TF/IDF reweighting of the counts obtained, thus rendering preprocessing as memory efficient as the actual computation of the kernel.

Multiclass Classification can sometimes lead to a very high dimensional feature vector even if the underlying feature map $x \to \phi(x)$ may be acceptable. For instance, for a bag-of-words representation of documents with 10^4 unique words and 10^3 classes this involves up to 10^7 coefficients to store the parameter vector directly when the $\phi(x, y) = e_y \otimes \phi(x)$, where $\otimes$ is the tensor product and e_y is a vector whose y-th entry is 1 and the rest are zero. The dimensionality of e_y is the number of classes.

Note that in the above case $\phi(x,y)$ corresponds to a sparse vector which has nonzero terms only in the part corresponding to e_y. That is, by using the joint index (i,y) with $\phi(x,y)_{(i,y')} = \phi_i(x)\delta_{y,y'}$ we may simply apply (6) to an extended index to obtain hashed versions of multiclass vectors. We have

$$\bar{\phi}_j(x,y) = \sum_{i \in \Im; h(i,y)=j} \phi_i(x). \tag{8}$$

In some cases it may be desirable to compute a compressed version of $\phi(x)$, that is, $\bar{\phi}(x)$ first and subsequently expand terms with y. In particular for strings this can be useful since it means that we need not parse x for every potential value of y. While this deteriorates the approximation in an additive fashion it can offer significant computational savings since all we need to do is permute a given feature vector as opposed to performing any summations.

Streams Some features of observations arrive as a stream. For instance, when performing estimation on graphs, we may obtain properties of the graph by using an MCMC sampler. The advantage is that we need not store the entire data stream but rather just use summary statistics obtained by hashing. Application for hash kernels on general graph data is future work.

4 Analysis

We show that the penalty we incur from using hashing to compress the number of coordinates only grows *logarithmically* with the number of objects and with the number of classes. While we are unable to obtain the excellent $O(\epsilon^{-1})$ rates offered by the count-min sketch, our approach retains the inner product property thus making hashing accessible to linear estimation.

Bias and Variance A first step in our analysis is to compute bias and variance of the approximation $\bar{\phi}(x)$ of $\phi(x)$. Whenever needed we will write $\bar{\phi}^h(x)$ and $\bar{k}^h(x,x')$ to make the dependence on the hash function h explicit. Using (6) we have

$$\begin{aligned}\bar{k}^h(x,x') &= \sum_j \sum_{i:h(i)=j} \phi_i(x) \sum_{i':h(i')=j} \phi'_i(x') \\ &= k(x,x') + \sum_{i,i':i \neq i'} \phi_i(x)\phi_{i'}(x')\delta_{h(i),h(i')}\end{aligned}$$

where δ is the Kronecker delta function. Taking the expectation with respect to the random choice of hash functions h we obtain the expected bias

$$\mathbf{E}_h[\bar{k}^h(x,x')] = \left(1 - \tfrac{1}{n}\right) k(x,x') + \tfrac{1}{n} \sum_i \phi_i(x) \sum_i \phi_{i'}(x')$$

Here we exploited the fact that for a random choice of hash functions the collision probability is $\frac{1}{n}$ uniformly over all pairs (i,j). Consequently $\bar{k}(x,x')$ is a biased estimator of the kernel matrix, with the bias decreasing inversely proportional to the number of hash bins.

The main change is a *rank-1* modification in the kernel matrix. Given the inherent high dimensionality of the estimation problem, a one dimensional change does not in general have a significant effect on generalization.

Straightforward (and tedious) calculation which is completely analogous to the above derivation leads to the following expression for the variance $\mathrm{Var}[\bar{k}^h(x,x')]$ of the hash kernel. Key in the derivation is our assumption that the family of hash functions we are dealing with is pairwise independent.

$$\tfrac{n-1}{n^2}\Big(k(x,x)k(x',x') + k^2(x,x') - 2\sum_i \phi_i^2(x)\phi_i^2(x')\Big)$$

As can be seen, the variance decreases $O(n^{-1})$ in the size of the values of the hash function. This means that we have an $O(n^{-\frac{1}{2}})$ convergence asymptotically to the expected value of the kernel.

Information Loss One of the key fears of using hashing in machine learning is that information is lost in the process which unavoidably restrains the best-possible prediction. For example, the well-known birthday paradox suggests that if the space that the hash function maps into has size n then if there are about $n^{0.5}$ features a collision is likely.

Redundancy in features is very helpful in avoiding information loss. This redundancy can be explicit or systemic such as might be expected with a bag-of-words or substring representation. In the following we analyze explicit redundancy where a feature is mapped to two or more values in the space of size n. This can be implemented with a hash function by (for example) appending the string $i \in \{1, \ldots, c\}$ to feature f and then computing the hash of $f \circ i$ for the i-th duplicate.

The essential observation with explicit feature duplication is that the information in a feature is only lost if all duplicates of the feature collide with another feature. Given this observation, it's unsurprising that increasing the size of n by a constant multiple c and duplicating features c times makes collisions with all features unlikely. It's perhaps more surprising that when keeping the size of n *constant* and duplicating features, the probability of losing the information in features due to collision can go *down*.

Theorem 1 *For a random function mapping l features duplicated c times into a space of size n, the probability (over the random function) that all features have at least one duplicate colliding with no other features is at least $1 - l[1 - (1 - c/n)^c + (lc/n)^c]$.*

To see the implications consider $l = 10^5$ and $n = 10^8$. Without duplication, a birthday paradox collision is virtually certain. However, if $c = 2$, the probability of a collision of all duplicates for any feature is bounded by about 0.404, and for $c = 3$ it drops to about 0.0117.

Proof The proof is essentially a counting argument with consideration of the fact that we are dealing with a hash *function* rather than a random variable.

Fix a feature f. We'll argue about the probability that all c duplicates of f collide with other features.

For feature duplicate i, let $h_i = h(f \circ i)$. The probability that $h_i = h(f' \circ i')$ for some other feature $f' \circ i'$ is bounded by $(l-1)c/n$ because the probability for each other mapping of a collision is $1/n$ by the assumption that h is a random function, and the union bound applied to the $(l-1)c$ mappings of other features yields $(l-1)c/n$. Note that we do not care about a collision of two duplicates of the same feature, because the feature value is preserved.

The probability that all duplicates $1 \le i \le c$ collide with another feature is bounded by $(lc/n)^c + 1 - (1 - c/n)^c$. To see this, let $c' \le c$ be the number of distinct duplicates of f after collisions. The probability of a collision with the first of these is bounded by $\frac{(l-1)c}{n}$. Conditioned on this collision, the probability of the next collision is at most $\frac{(l-1)c-1}{n-1}$, where 1 is subtracted because the first location is fixed. Similarly, for the ith duplicate, the probability is $\frac{(l-1)c-(i-1)}{n-(i-1)}$. We can upper bound each term as $\frac{lc}{n}$, implying the probability of all c' duplicates colliding with other features is at most $(lc/n)^{c'}$. The probability that $c' = c$ is the probability that none of the duplicates of f collide, which is $\frac{(n-1)!}{n^c(n-c-1)!} \ge ((n-c)/n)^c$. If we pessimistically assume that $c' < c$ implies that every duplicate collides with another feature, then

$$\begin{aligned} \mathrm{P}(\text{coll}) &\le \mathrm{P}(\text{coll}|c' = c)\,\mathrm{P}(c' = c) + \mathrm{P}(c' \neq c) \\ &\le (lc/n)^c + 1 - ((l-c)/l)^c. \end{aligned}$$

Simplification gives $(lc/n)^c + 1 - (1 - c/n)^c$ as claimed. Taking a union bound over all l features, we get that the probability any feature has all duplicates collide is bounded by $l[1-(1-c/n)^c+(lc/n)^c]$. ■

Rate of Convergence As a first step note that any convergence bound only depends *logarithmically* on the size of the kernel matrix.

Theorem 2 *Assume that the probability of deviation between the hash kernel and its expected value is bounded by an exponential inequality via*

$$\mathrm{P}\left[\left|\bar{k}^h(x,x') - \mathbf{E}_h\left[\bar{k}^h(x,x')\right]\right| > \epsilon\right] \le c\exp(-c'\epsilon^2 n)$$

for some constants c, c' depending on the size of the hash and the kernel used. In this case the error ϵ arising from ensuring the above inequality for m observations and M classes (for a joint feature map $\phi(x,y)$) is bounded by (with $c'' := -\log c - 2\log 2$)

$$\epsilon \le \sqrt{(2\log(m+1) + 2\log(M+1) - \log\delta - c'')/c'}.$$

Proof [sketch only] Apply the union bound to the kernel matrix of size $(mM)^2$, that is, to all $m(m+1)M(M+1)/4$ unique elements. Taking logs on both sides and solving for ϵ yields the result. ■

To obtain an exponential bound on the probability of deviation note that while each coordinate of the hashed feature is identically distributed, the set of features is only exchangeable but not independent. Although the effect is small, this precludes a direct application of Chernoff bounds.

Using the variance calculation, Chebyshev's inequality gives bounds on the probability of large deviation by ϵ of the form $O(n^{-1}\epsilon^{-2})$, albeit not an exponential inequality. Bounds for exchangeable random variables, in particular (Chatterjee, 2005, Theorem 3.3), can be used to obtain an exponential inequality in ϵ of the form required by Theorem 2, albeit without providing the desired dependency on n. We conjecture that a bound providing both scaling behavior exists (and our conjecture is supported by our experimental findings). This is subject of future work.

5 Experiments

To test the efficacy of our approach we applied hashing to the following problems: first we used it for classification on the Reuters RCV1 dataset as it has a relatively large feature dimensionality. Secondly, we applied it to the DMOZ ontology of topics of webpages[1] where the number of topics is high.

5.1 Reuters Articles Categorization

We use the Reuters RCV1 binary classification dataset (Lewis et al., 2004). 781,265 articles are used for training by stochastic gradient descent (SGD) and 23,149 articles are used for testing. Conventionally one would build a bag of words representation first and calculate exact term frequency / inverse document frequency (TF-IDF) counts from the contents of each article as features. The problem is that the TF calculation needs to maintain a very large dictionary throughout the whole process. Moreover, it is impossible to extract

[1] Dmoz L2 denotes non-parent topic data in the top 2 levels of the topic tree and Dmoz L3 denotes non-parent topic data in the top 3 levels of the topic tree.

Datasets	#Train	#Test	#Label
RCV1	781,265	23,149	2
Dmoz L2	4,466,703	138,146	575
Dmoz L3	4,460,273	137,924	7,100

Table 1: Text datasets. #X denotes the number of observations in X.

Algorithm	Pre	TrainTest	Error %
BSGD	303.60s	10.38s	6.02
VW	303.60s	510.35s	5.39
VWC	303.60s	5.15s	5.39
HK	0	25.16s	5.60

Table 2: Runtime and Error on RCV1. BSGD: Bottou's SGD, VW: Vowpal Wabbit without cache, VWC: Vowpal Wabbit using cache file, HK: Hash kernel; Pre: preprocessing time, TrainTest: time to load data, train and test the model; Error: misclassification rate

bits	#unique	Collision %	Error %
24	285614	0.82	5.586
22	278238	3.38	5.655
20	251910	12.52	5.594
18	174776	39.31	5.655
16	64758	77.51	5.763
14	16383	94.31	6.096

Table 3: Influence of the hash size on Reuters (RCV1) on collisions (reported for both training and test set combined) and error rates. Note that there is no noticeable performance degradation even for a 40% collision.

features online since the entire vocabulary dictionary is usually unobserved during training. Another disadvantage is that calculating exact IDF requires us to preprocess all articles in a first pass. This is not possible as articles such as news may vary daily.

However, it suffices to compute TF and IDF approximately as follows: using hash features, we no longer require building the bag of words. Every word produces a hash key which is the dimension index of the word. The frequency is recorded in the dimension index of its hash key. Therefore, every article has a frequency count vector as TF. This TF is a much denser vector which requires no knowledge of the vocabulary. IDF can be approximated by scanning a smaller *part* of the training set.

A quantile-quantile plot in Figure 1 shows that this approximation is justified — the dependency between the statistics on the subset (200k articles) and the full training set (800k articles) is perfectly linear.

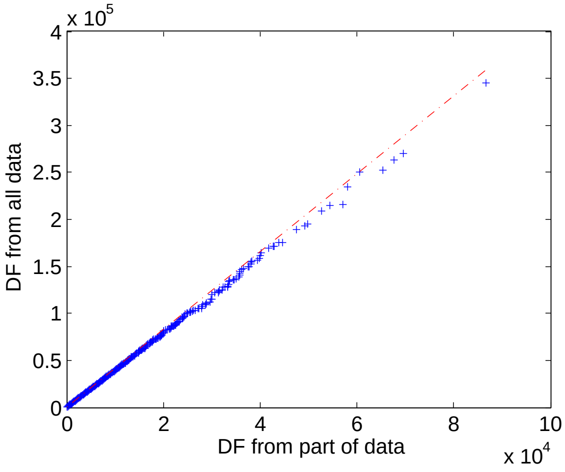

Figure 1: Quantile-quantile plot of the DF counts computed on a subset (200k documents) and the full dataset (800k documents). DF(t) is the number of documents in a collection containing word t.

We compare the hash kernel with Leon Bottou's Stochastic Gradient Descent SVM[2] (BSGD) and Vowpal Wabbit[3] (VW). Our hash scheme is generating features online. BSGD is generating features offline and learning them online. VW uses BSGD's preprocessed features and creates further features online. Caching speeds up VW considerably. However, it requires one run of the original VW code for this purpose. We compare all three algorithms on RCV1 in Table 2. As can be seen, the preprocessing time of BSGD and VW is considerably longer compared to the time for training and testing, due to the TF-IDF calculation which is carried out offline. Hashing avoids this and we generate features on the fly as we are learning the model. For a fair comparison, we measure the time for feature loading, training and testing together. As can be seen, the speed of online feature generation is considerable compared to disk access.

Furthermore, we investigate the influence of hash size (i.e. the number of hash bins) on the misclassification rate. As can be seen in Table 3, when the hash size decreases, the collision and the error rate increase. In particular, a 24 bit hash causes almost no collisions. Nonetheless, an 18 bit hash which has almost 40% collisions performs equally well on the problem. This leads to rather memory-efficient implementations.

5.2 Dmoz Websites Multiclass Classification

In a second experiment we perform topic categorization using the topic ontology DMOZ. The task is to recognize the topic of websites given the short descriptions provided on the webpages. To simplify things we

[2] http://leon.bottou.org/projects/sgd

[3] http://hunch.net/~vw/

	HLF 28bit		**HLF 24bit**		**HF**		**no hash**	**U base**	**P base**
	error	memory	error	memory	error	memory	memory	error	error
L2	30.12	2G	30.71	0.125G	31.28	2.25G (19bit)	7.85G	99.83	85.05
L3	52.1	2G	53.36	0.125G	51.47	1.73G (15bit)	96.95G	99.99	86.83

Table 4: Misclassification and memory footprint of hashing and baseline methods on DMOZ. HLF: joint hashing of labels and features. HF: hash features only. no hash: direct model (not implemented as too large, hence only memory estimates — we have 1,832,704 unique words). U base: baseline of uniform classifier. P base: baseline of majority vote. Note: the memory footprint in HLF is essentially independent of the number of classes used.

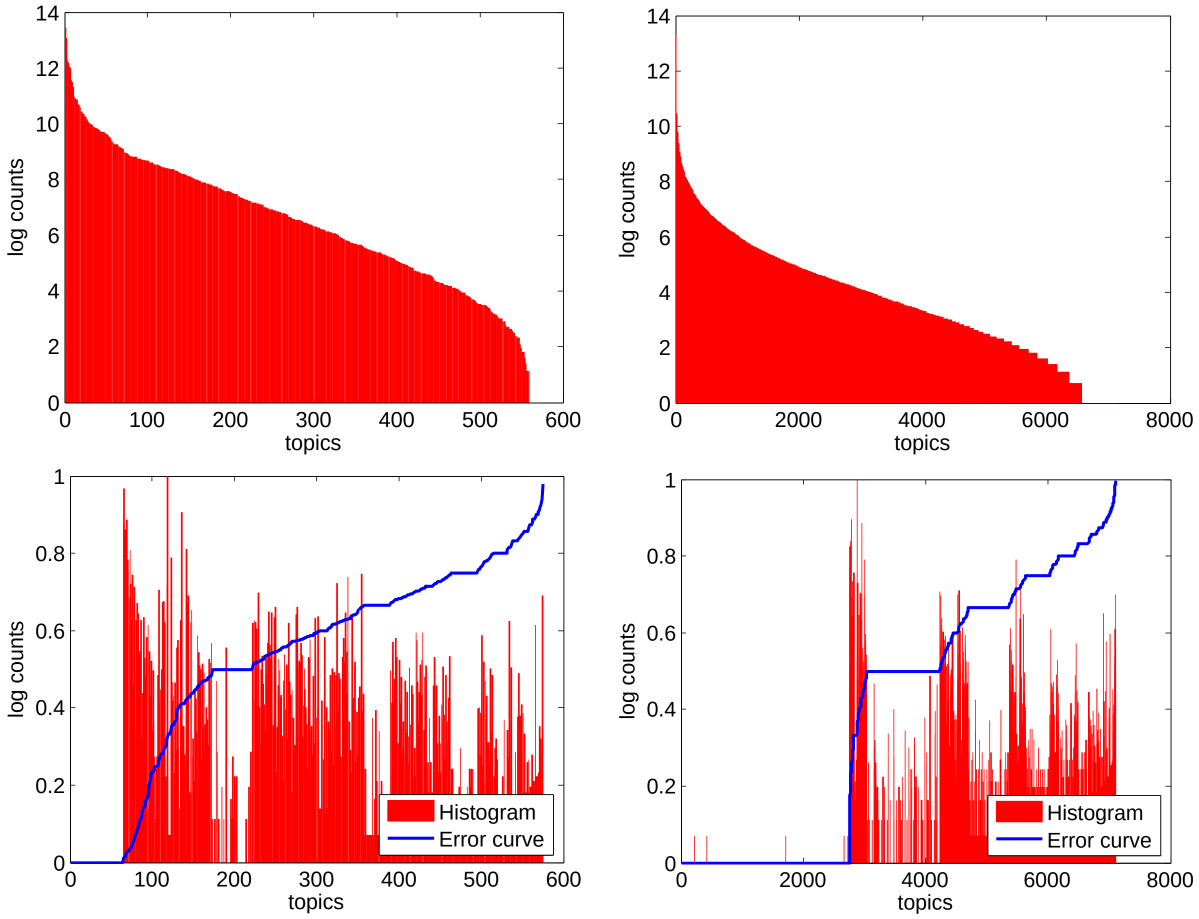

Figure 2: Left: results on L2, Right: results on L3; Top: frequency counts for topics as reported on the training set (the test set distribution is virtually identical). We see an exponential decay in counts. Bottom: log-counts and error probabilities on the test set. Note that the error is reasonably evenly distributed among the size of the classes (besides a number of near empty classes which are learned perfectly).

categorize only the leaf nodes (level 2: L2 or level 3: L3) as a flat classifier (the hierarchy could be easily taken into account by adding hashed features for each part of the path in the tree). This leaves us with 575 leaf topics on L2 and with 7100 leaf topics on L3.

Conventionally, assuming M classes and l features, training M different parameter vectors w requires $O(Ml)$ storage. This is infeasible for massively multiclass applications. However, by hashing data and labels jointly we are able to obtain an efficient joint representation which makes the implementation computationally possible.

As can be seen in Table 4 joint hashing of features and labels outperforms all other approaches and in many cases is necessary to make large multiclass categorization computationally feasible at all (competing methods run out of memory). In particular, hashing features only produces slightly worse results than joint hashing of labels and features. This is likely due to the increased collision rate: we need to use a smaller hash to store the class dependent weight vectors explicitly.

Next we investigate whether such good misclassification rate is obtained by predicting well only on a few dominant topics. We reorder the topic histogram in accordance to ascending error rate. Figure 2 shows that the hash kernel does very well on the first one hundred topics. They correspond to easy categories such as language related sets "World/Italiano","World/Japanese","World/Deutsch".

6 Discussion

In this paper we showed that hashing is a computationally attractive technique which allows one to approximate kernels for very high dimensional settings efficiently by means of a sparse projection into a lower dimensional space. In particular for multiclass categorization this makes all the difference in terms of being able to implement problems with thousands of classes in practice on large amounts of data and features. How to apply hash kernels for data which have more complex structure such as general graph and how to get a unbiased estimator of the kernel are future work.

References

C. Berg, J. P. R. Christensen, and P. Ressel. *Harmonic Analysis on Semigroups*. Springer, New York, 1984.

C. J. C. Burges and B. Schölkopf. Improving the accuracy and speed of support vector learning machines. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 375–381, Cambridge, MA, 1997. MIT Press.

S. Chatterjee. *Concentration Inequalities with Exchangeable Pairs*. PhD thesis, Stanford University, 2005.

G. Cormode and M. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. In *LATIN: Latin American Symposium on Theoretical Informatics*, 2004.

O. Dekel, S. Shalev-Shwartz, and Y. Singer. The Forgetron: A kernel-based perceptron on a fixed budget. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*, Cambridge, MA, 2006. MIT Press.

S. Fine and K. Scheinberg. Efficient SVM training using low-rank kernel representations. *JMLR*, 2:243–264, Dec 2001.

K. Ganchev and M. Dredze. Small statistical models by random feature mixing. In *workshop on Mobile NLP at ACL*, 2008.

P. Indyk and R. Motawani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the* 30^{th} *Symposium on Theory of Computing*, pages 604–613, 1998.

L. Kontorovich. A universal kernel for learning regular languages. In *Machine Learning in Graphs*, 2007.

J. Langford, L. Li, and A. Strehl. Vowpal wabbit online learning project (technical report). Technical report, 2007.

D. Lewis, Y. Yang, T. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *The Journal of Machine Learning Research*, 5:361–397, 2004.

P. Li, K. Church, and T. Hastie. Conditional random sampling: A sketch-based sampling technique for sparse data. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 873–880. MIT Press, Cambridge, MA, 2007.

O. L. Mangasarian. Generalized support vector machines. Technical report, University of Wisconsin, Computer Sciences Department, Madison, 1998.

N. Przulj. Biological network comparison using graphlet degree distribution. *Bioinformatics*, 23(2): e177–e183, Jan 2007.

A. Rahimi and B. Recht. Random features for large-scale kernel machines. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*. MIT Press, Cambridge, MA, 2008.

B. Schölkopf. *Support Vector Learning*. R. Oldenbourg Verlag, Munich, 1997. Download: http://www.kernel-machines.org.

C. H. Teo and S. V. N. Vishwanathan. Fast and space efficient string kernels using suffix arrays. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pages 929–936, New York, NY, USA, 2006. ACM Press. ISBN 1-59593-383-2. doi: http://doi.acm.org/10.1145/1143844.1143961.

K. Tsuda, T. Kin, and K. Asai. Marginalized kernels for biological sequences. *Bioinformatics*, 18 (Suppl. 2):S268–S275, 2002.

S. V. N. Vishwanathan, A. J. Smola, and R. Vidal. Binet-Cauchy kernels on dynamical systems and its application to the analysis of dynamic scenes. *International Journal of Computer Vision*, 73(1):95–119, 2007.

C. Watkins. Dynamic alignment kernels. In A. J. Smola, P. L. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 39–50, Cambridge, MA, 2000. MIT Press.