Journal of Machine Learning Research 11 (2009) xxx-xxx Submitted 02/09; Published 06/09

Hash Kernels for Structured Data

Qinfeng Shi QINFENG.SHI@ANU.EDU.AU
James Petterson JPETTERSON@GMAIL.COM
NICTA and CECS of Australian National University
Canberra ACT 2601,Australia

Gideon Dror GIDEON@MTA.AC.IL
Division of Computer Science
Academic College of Tel-Aviv-Yaffo, Israel

John Langford JL@HUNCH.NET
Alex Smola ALEX.SMOLA@GMAIL.COM
Yahoo! Research
New York, NY and Santa Clara, CA, USA

S.V.N. Vishwanathan VISHY@MAIL.RSISE.ANU.EDU.AU
Department of Statistics
Purdue University, IN, USA

Editor: Soeren Sonnenburg, Vojtech Franc, Elad Yom-Tov and Michele Sebag

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.

Keywords: Hashing, Stream, String Kernel, Graphlet Kernel, Multiclass Classification

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 $\bar{\phi} : \mathcal{X} \to \bar{\mathcal{F}}$

$$\bar{k}(x, x') = \langle \bar{\phi}(x), \bar{\phi}(x') \rangle \text{ often } \bar{\phi}(x) = M\phi(x). \tag{2}$$

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 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 which is suitable for data with simple structure such as strings. An analysis follows in Section 4. And we propose a graphlet kernel which generalizes hash kernels to data with general structure — graphs and discuss a MCMC sampler in Section 5. Finally we conclude with experiments in Section 6.

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 $\bar{\phi}(x) = n^{-\frac{1}{2}} \left(\phi_{c_1}(x), \ldots, \phi_{c_n}(x)\right)$ in the sense that their resulting kernel is similar. 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. (2007b) 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 our 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 of 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 $\mathrm{P}(c)$ 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. For example, $v_i \sim \mathcal{N}(0, I)$ is sufficient, where I is an identity matrix. This allows one to obtain Chernoff bounds and $O(\epsilon^{-2}\log m)$ rates of approximation, where m is the number of instances. The main cost is to store v_i and perform the $O(nm)$ multiply-adds, thus rendering this approach too expensive as a preprocessing step in many applications.

Achlioptas (2003) proposes a random projection approach that uses symmetric random variables to project the original feature onto a lower dimension feature space. This operation is simple and faster and the author shows it does not sacrifice the quality of the embedding. Moreover, it can be directly applied to online learning tasks. Unfortunately, the projection remains dense resulting in relatively poor computational and space performance in our experiments.

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, where $1 - \delta$ is the confidence.

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.

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 the one discussed here.

Hash kernel on strings Shi et al. (2009) propose a hash kernel to deal with the issue of computational efficiency by a very simple algorithm: high-dimensional vectors are compressed by adding up all coordinates which have the same hash value — one only needs to perform as many calculations as there are nonzero terms in the vector. The hash kernel can jointly hash both label and features, thus the memory footprint is essentially independent of the number of classes used.

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 graphics cards).

Kernel Approximation 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 $\phi(x)$ 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:

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

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

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

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 pre-processing 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 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

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

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.

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 $\mathrm{Var}_\mathrm{h}[\bar{k}^h(x, x')]$ of the hash kernel:

$$\mathrm{Var}_h[\bar{k}^h(x, x')] = \tfrac{n-1}{n^2}\Big(k(x, x)k(x', x') + k^2(x, x') - 2\sum_i \phi_i^2(x)\phi_i^2(x')\Big)$$

Key in the derivation is our assumption that the family of hash functions we are dealing with is pairwise independent.

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 hash collisions harm performance. For example, the well-known birthday paradox shows that if the hash function maps into a space of size n then with $O(n^{\frac{1}{2}})$ features a collision is likely. When a collision occurs, information is lost, which may reduce the achievable performance for a predictor.

Definition 1 *(Information Loss) A hash function h causes information loss on a distribution D with a loss function L if the expected minimum loss before hashing is less than the expected minimum loss after hashing:*

$$\min_f \mathop{E}_{(x,y)\sim D}[L(f(x), y))] < \min_g \mathop{E}_{(x,y)\sim D}[L(g(h(x)), y))]$$

Redundancy in features is very helpful in avoiding information loss. The 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 is that the information in a feature is only lost if all duplicates of the feature collide with other features. 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 information loss can go *down*.

Theorem 2 *For a random function mapping l features duplicated c times into a space of size n, for all loss functions L and and distributions D on n features, the probability (over the random function) of no information loss is:*

$$\geq 1 - l[1 - (1 - c/n)^c + (lc/n)^c]$$

.

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 information loss 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. It is structurally similar to the proof for a Bloom filter (Bloom, 1970), because the essential question we address is: "What is a lower bound on the probability that all features have one duplicate not colliding with any other feature?"

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 $1 \leq i \leq c$ collide with another feature is bounded by $(lc/n)^c + 1 - (1 - c/n)^c$. To see this, let $c' \leq c$ 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 $\frac{(n-1)!}{n^c(n-c-1)!} \geq ((n-c)/n)^c$. If we pessimistically assume that $c' < c$ implies that every duplicate collides with another feature, then

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

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

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

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 $c'' := -\log c - 2\log 2$)

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

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 3, 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.

Generalization Bound The hash kernel approximates the original kernel by big storage and computation saving. An interesting question is whether the generalization bound on the hash kernel will be **much worse** than the bound obtained on the original kernel.

Theorem 4 (Generalization Bound) *For binary class SVM, let $\bar{K} = \langle\bar{\phi}(x), \bar{\phi}(x')\rangle, K = \langle\phi(x), \phi(x')\rangle$ be the hash kernel matrix and original kernel matrix, let the training set $S = \{(x^1, y^1), \ldots, (x^M, y^M)\}$ be drawn independently according to a probability distribution $\mathcal{D}$ from $\mathcal{X} \times \mathcal{Y}$ and fix $\delta \in (0, 1)$. Assume there exists $b \leq 0$ such that $b \geq tr(\bar{K}) - tr(K)$. Then with probability at least $1 - \delta$ over samples of size M we have*

$$P(y \neq f(x, w)) \leq \frac{1}{M}\sum_{m=1}^{M}\xi_m + \frac{4}{M}\sqrt{tr(\bar{K})} + 3\sqrt{\frac{ln(\frac{2}{\delta})}{2M}}, \tag{9}$$

$$\leq \frac{1}{M}\sum_{m=1}^{M}\xi_m + \frac{4}{M}\sqrt{tr(K)} + \frac{4}{M}\sqrt{b} + 3\sqrt{\frac{ln(\frac{2}{\delta})}{2M}}, \tag{10}$$

where $\xi_m = \max\{0, \triangle(y^m, y^m_*) - \langle w, \bar{\phi}(x^m, y^m)\rangle + \langle w, \bar{\phi}(x^m, y^m_*)\rangle\}$

Proof [sketch] Using Rademacher complexity we can get inequality 9. Using the inequality $tr(\bar{K}) \leq b + tr(K) \leq (\sqrt{b} + \sqrt{tr(K)})^2$ in inequality 9 gives inequality 10. So the theorem holds. ■

The approximation quality depends on the both the feature and the hash collision. From the definition of hash kernel (see eq. 6), the feature entries with the same hash value will add up and map to a new feature entry indexed by the hash value. The higher collision of the hash has, the more entries will be added up. If the feature is sparse, the added up entries are mostly zeros. So the difference of the maximum and the sum of the entries is reasonably small. In this case, the hash kernel gives a good approximation of the original kernel, so b is reasonably small. Thus the generalization error does not increase much as the collision increases. This is verified in the experiment section in Table 4 — increasing the collision rate from 0.82% to 94.31% only slightly worsens the test error (from 6.096% to 5.586%).

Moreover, Theorem 4 shows us the generalization bounds on the hash kernel and the original kernel only differ by $O(M^{-1})$. This means that when our dataset is large, the difference can be ignored.

5. Graphlet Kernels

Denote by G a graph with vertices $V(G)$ and edges $E(G)$. Several methods have been proposed to perform classification on such graphs. Most recently, Przulj (2007) proposed to use the distribution over graphlets, i.e. subgraphs, as a characteristic property of the graph. Unfortunately, brute force evaluation does not allow calculation of the statistics for graphlets of size more than 5, since the cost for exact computation scales exponentially in the graphlet size.

In the following we show that sampling and hashing can be used to make the analysis of larger subgraphs tractable in practice. For this denote by $S \subseteq G$ an induced subgraph of G, obtained by restricting ourselves to only $V(S) \subseteq V(G)$ vertices of G and let $\#_S(G)$ be the number of times S occurs in G. This suggests that the feature map $G \to \phi(G)$, where $\phi_S(G) = \#_S(G)$ will induce a useful kernel: adding or removing an edge (i, j) only changes the properties of the subgraphs using the pair (i, j) as part of their vertices.

5.1 Counting and Sampling

Depending on the application, the distribution over the counts of subgraphs may be significantly skewed. For instance, in sparse graphs we expect the fully disconnected subgraphs to be considerably overrepresented. Likewise, whenever we are dealing with almost complete graphs, the distribution may be skewed towards the other end (i.e. most subgraphs will be complete). To deal with this, we impose weights $\beta(k)$ on subgraphs containing k edges $|E(S)|$.

To deal with the computational complexity issue simultaneously with the issue of reweighting the graphs S we simply replace explicit counting with sampling from the distribution

$$\mathrm{P}(S|G) = c(G)\beta(|E(S)|) \tag{11}$$

where $c(G)$ is a normalization constant. Samples from $\mathrm{P}(S|G)$ can be obtained by a Markov-Chain Monte Carlo approach.

Lemma 5 *The following MCMC sampling procedure has the stationary distribution (11).*

1. *Choose a random vertex, say i, of S uniformly.*
2. *Add a vertex j from $G\backslash S_i$ to S_i with probability $c(S_i, G)\beta(|E(S_{ij})|)$.*

Here S_i denotes the subgraph obtained by removing vertex i from S, and S_{ij} is the result of adding vertex j to S_i.

Note that sampling over j is easy: all vertices of G which do not share an edge with $S\backslash i$ occur with the same probability. All others depend only on the number of joining edges. This allows for easy computation of the normalization $c(S_i, G)$.

Proof We may encode the sampling rule via

$$T(S_{ij}|S, G) = \frac{1}{k}c(S_i, G)\beta(|E(S_{ij})|) \tag{12}$$

where $c(S_i, G)$ is a suitable normalization constant. Next we show that T satisfies the balance equations and therefore can be used as a proposal distribution with acceptance probability 1.

$$\begin{aligned}&\frac{T(S_{ij}|S,G)\,\mathrm{P}(S)}{T(S|S_{ij},G)\,\mathrm{P}(S_{ij})}\\ =&\frac{k^{-1}c(S_i,G)\beta(|E(S_{ij})|)c(G)\beta(|E(S)|)}{k^{-1}c(S_{ij,j},G)\beta(|E(S_{ij,ji})|)c(G)\beta(|E(S_{ij})|)} = 1\end{aligned}$$

This follows since $S_{ij,j} = S_i$ and likewise $S_{ij,ji} = S$. That is, adding and removing the same vertex leaves a graph unchanged. ■

In summary, we obtain an algorithm which will readily draw samples S from $\mathrm{P}(S|G)$ to characterize G.

5.2 Dependent Random Variables

The problem with sampling from a MCMC procedure is that the random variables are *dependent* on each other. This means that we cannot simply appeal to Chernoff bounds when it comes to averaging. Before discussing hashing we briefly discuss averages of dependent random variables:

Definition 6 (Bernoulli Mixing) *Denote by Z a stochastic process indexed by $t \in \mathbb{Z}$ with probability measure P and let Σ_n be the σ-algebra on Z_t with $t \in \mathbb{Z}\backslash 1, \ldots, n-1$. Moreover, denote by P_- and P_+ the probability measures on the negative and positive indices t respectively. The mixing coefficient β is*

$$\beta(n, \mathrm{P}_X) := \sup_{A \in \Sigma_n} \Big|\mathrm{P}(A) - \mathrm{P}_- \times \mathrm{P}_+(A)\Big|. \tag{13}$$

If $lim_{n\to\infty}\beta(n, \mathrm{P}_z) = 0$ we call Z to be β-mixing.

That is, $\beta(n, \mathrm{P}_X)$ measures how much dependence a sequence has when cutting out a segment of length n. Nobel and Dembo (1993) show how such mixing processes can be related to iid observations.

Theorem 7 *Assume that P is β-mixing. Denote by P^* the product measure obtained from $\ldots \mathrm{P}_t \times \mathrm{P}_{t+1} \ldots$ Moreover, denote by $\Sigma_{l,n}$ the σ-algebra on $Z_n, Z_{2n}, \ldots, Z_{ln}$. Then the following holds:*

$$\sup_{A \in \Sigma_{l,n}} |\,\mathrm{P}(A) - \mathrm{P}^*(A)| \leq l\beta(n, \mathrm{P}). \tag{14}$$

This allows us to obtain bounds for expectations of variables drawn from P rather than P^*.

Theorem 8 *Let* P *be a distribution over a domain* $\mathcal{X}$ *and denote by* $\phi : \mathcal{X} \to \mathcal{H}$ *a feature map into a Hilbert Space with* $\langle \phi(x), \phi(x') \rangle \in [0, 1]$. *Moreover, assume that there is a* β*-mixing MCMC sampler of* P *with distribution* P^{MC} *from which we draw* l *observations* x_{in} *with an interleave of* n *rather than sampling from* P *directly. Averages with respect to* P^{MC} *satisfy the following with probability at least* $1 - \delta$*:*

$$\left\| \mathop{\mathbf{E}}_{x \sim \mathrm{P}(x)} [\phi(x)] - \frac{1}{l} \sum_{i=1}^{l} \phi(x_{in}) \right\| \leq l\beta(n, \mathrm{P}^{\mathrm{MC}}) + \frac{2 + \sqrt{\log \frac{2}{\delta}}}{\sqrt{l}}$$

Proof Theorem 7, the bound on $\|\phi(x)\|$, and the triangle inequality imply that the expectations with respect to P^{MC} and P^* only differ by $l\beta$. This establishes the first term of the bound. The second term is given by a uniform convergence result in Hilbert Spaces from Altun and Smola (2006). ■

Hence, sampling from a MCMC sampler for the purpose of approximating inner products is sound, provided that we only take sufficiently independent samples (i.e. a large enough n) into account. The translation of Theorem 8 into bounds on inner products is straightforward, since

$$\begin{aligned} & |\langle x, y \rangle - \langle x', y' \rangle| \\ & \leq \|x - x'\| \|y\| + \|y - y'\| \|x\| + \|x - x'\| \|y - y'\| . \end{aligned} \tag{15}$$

5.3 Hashing and Subgraph Isomorphism

Sampling from the distribution over subgraphs $S \in G$ has two serious problems in practice which we will address in the following: firstly, there are several graphs which are isomorphic to each other. This needs to be addressed with a graph isomorphism tester, such as Nauty McKay (1984). For graphs up to size 12 this is a very effective method. Nauty works by constructing a lookup table to match isomorphic objects.

However, even after the graph isomorphism mapping we are still left with a sizable number of distinct objects. This is where a hash map on data streams comes in handy. It obviates the need to store any intermediate results, such as the graphs S or their unique representations obtained from Nauty. Finally, we combine the convergence bounds from Theorem 8 with the guarantees available for hash kernels to obtain the approximate graph kernel.

Note that the two randomizations have very different purposes: the sampling over graphlets is done as a way to approximate the *extraction* of features whereas the compression via hashing is carried out to ensure that the representation is computationally efficient.

6. 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

Datasets	**#Train**	**#Test**	**#Labels**
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	87.63s	5.39
VWC	303.60s	5.15s	5.39
HK	0s	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 with feature dimension 2^{20}. Pre: preprocessing time. TrainTest: time to load data, train and test the model. Error: misclassification rate. Apart from the efficacy of hashing operation itself, the gain of speed is also due to a multi-core implementation — hash kernel uses 4-cores to access the disc for online hash feature generation. For learning and testing evaluation, all algorithms use single-core.

dimensionality. Secondly, we applied it to the DMOZ ontology of topics of webpages[1] where the number of topics is high. The third experiment — Biochemistry and Bioinformatics Graph Classification uses our hashing scheme, which makes comparing all possible subgraph pairs tractable, to compare graphs Vishwanathan et al. (2007a). On publicly available datasets like MUTAG and PTC as well as on the biologically inspired dataset DD used by Vishwanathan et al. (2007a), our method achieves the best known accuracy.

In both RCV1 and Dmoz, we use linear kernel SVM with stochastic gradient descent (SGD) as the workhorse. We apply our hash kernels and random projection (Achlioptas, 2003) to the SGD linear SVM. We don't apply the approach in Rahimi and Recht (2008) since it requires a shift-invariant kernel $k(x, y) = k(x - y)$, such as RBF kernel, which is not applicable in this case. In the third experiment, existing randomization approaches are not applicable since enumerating all possible subgraphs is intractable. Instead we compare hash kernel with existing graph kernels: random walk kernel, shortest path kernel and graphlet kernel (see Borgwardt et al. (2007)).

6.1 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.

Algorithm	Dim	Pre	TrainTest	orgTrainSize	newTrainSize	Error %
RP	2^8	748.30s	210.23s	423.29Mb	1393.65Mb	29.35%
	2^9	1079.30s	393.46s	423.29Mb	2862.90Mb	25.08%
	2^{10}	1717.30s	860.95s	423.29Mb	5858.48Mb	19.86%
HK	2^8	0s	22.82s	NA	NA	17.00%
	2^9	0s	24.19s	NA	NA	12.32%
	2^{10}	0s	24.41s	NA	NA	9.93%

Table 3: Hash kernel vs. random projections with various feature dimensionalities on RCV1. RP: random projections in Achlioptas (2003). HK: hash kernel. Dim: dimension of the new features. Pre: preprocessing time. TrainTest: time to load data, train and test the model. orgTrainSize: compressed original training feature file size. newTrainSize: compressed new training feature file size. Error: misclassification rate. NA: not applicable. In hash kernel there is no preprocess step, so there is no original/new feature files. Features for hash kernel are built up online via accessing the string on disc. The disc access time is taken into account in **TrainTest**. Note that the TrainTest for random projection time increases as the new feature dimension increases, whereas for hash kernel the TrainTest is almost independent of the feature dimensionality.

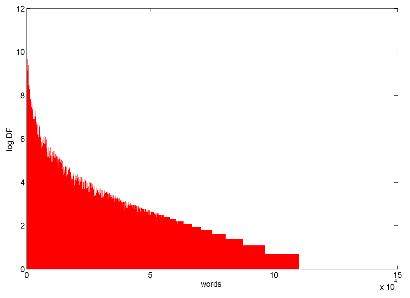
Dim	#Unique	Collision %	Error %
2^{24}	285614	0.82	5.586
2^{22}	278238	3.38	5.655
2^{20}	251910	12.52	5.594
2^{18}	174776	39.31	5.655
2^{16}	64758	77.51	5.763
2^{14}	16383	94.31	6.096

Table 4: Influence of new dimension on Reuters (RCV1) on collision rates (reported for both training and test set combined) and error rates. Note that there is no noticeable performance degradation even for a 40% collision rate.

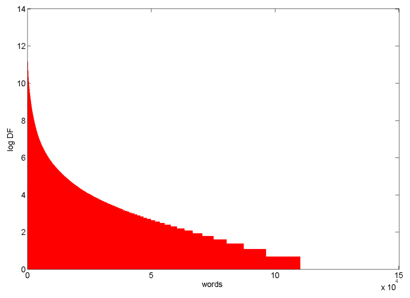
features online since the entire vocabulary dictionary is usually unobserved during training. Another disadvantage is that calculating exact IDF requires us to preprocess all articles in a first pass. This is not possible as articles such as news may vary daily.

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.

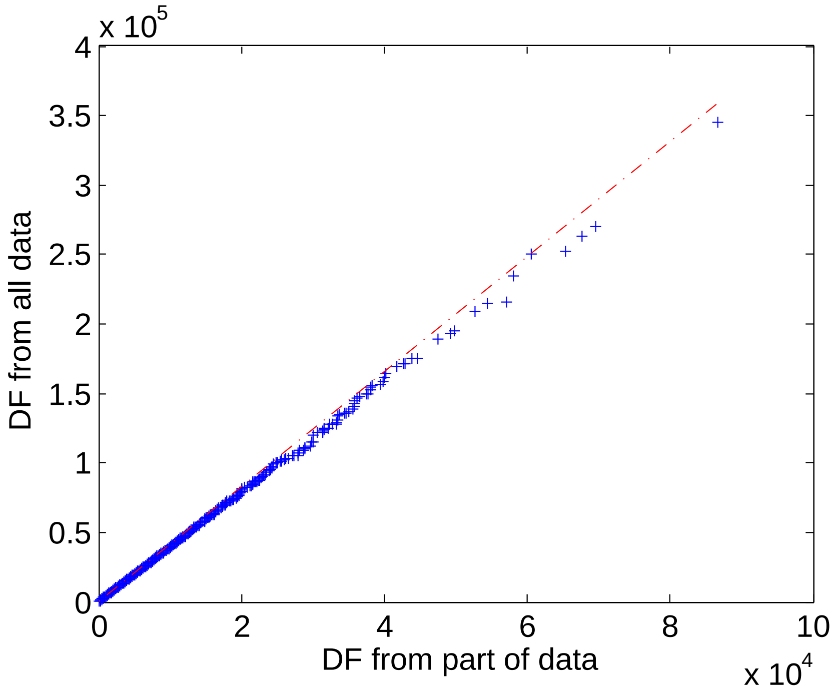
A quantile-quantile plot in Figure 1 shows that this approximation is justified — the dependency between the statistics on the subset (200k articles) and the full training set (800k articles) is perfectly linear.



(a) DF from part of data



(b) DF from all data



(c) Quantile-Quantile plot

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), Vowpal Wabbit (Langford et al., 2007) (VW) and Random Projections (RP) Achlioptas (2003). 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. RP uses BSGD's preprocessed features and then creates the new projected lower dimension features. Then it uses BSGD for learning and testing. We compare these algorithms on RCV1 in Table 2. Table 2. RP is

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

not included in this table because it would be intractable to run it with the same feature dimensionality as HK for a fair comparison. 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. For a fair comparison, we measure the time for feature loading, training and testing together. It can also be seen that the speed of online feature generation is considerable compared to disk access. Table 2 shows that the test errors for hash kernel, BSGD and VW are competitive.

In table 3 we compare hash kernel to RP with different feature dimensions. As we can see, the error reduces as the new feature dimension increases. However, the error of hash kernel is always much smaller (by about 10%) than RP given the same new dimension. An interesting thing is that the new feature file created after applying RP is much bigger than the original one. This is because the projection maps the original sparse feature to a dense feature. For example, when the feature dimension is 2^{10}, the compressed new feature file size is already 5.8G. Hash kernel is much more efficient than RP in terms of speed, since to compute a hash feature one requires only $O(d_{nz})$ hashing operations, where d_{nz} is the number of non-zero entries. To compute a RP feature one requires $O(dn)$ operations, where d is the original feature dimension and and n is the new feature dimension. And with RP the new feature is always dense even when n is big, which further increases the learning and testing runtime. When $d_{nz} \ll d$ such as in text process, the difference is significant. This is verified in our experiment (see in Table 3). For example, hash kernel (including **Pre** and **TrainTest**) with 2^{10} feature size is over 100 times faster than RP.

Furthermore, we investigate the influence of the new feature dimension on the misclassification rate. As can be seen in Table 4, when the feature dimension decreases, the collision and the error rate increase. In particular, a 2^{24} dimension causes almost no collisions. Nonetheless, a 2^{18} dimension which has almost 40% collisions performs equally well on the problem. This leads to rather memory-efficient implementations.

6.2 Dmoz Websites Multiclass Classification

In a second experiment we perform topic categorization using the DMOZ topic ontology. The task is to recognize the topic of websites given the short descriptions provided on the webpages. To simplify things we categorize only the leaf nodes (Top two levels: L2 or Top three levels: 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 5 joint hashing of features and labels is very attractive in items of memory usage and in many cases is necessary to make large multiclass categorization computationally feasible at all (naive online SVM ran out of memory). In particular, hashing features only produces worse results than joint hashing of labels and features. This is likely due to the increased collision rate: we need to use a smaller feature dimension to store the class dependent weight vectors explicitly.

	HLF (2^{28})		**HLF** (2^{24})		**HF**		**no hash**	**U base**	**P base**
	error	mem	error	mem	error	mem	mem	error	error
L2	30.12	2G	30.71	0.125G	31.28	2.25G (2^{19})	7.85G	99.83	85.05
L3	52.10	2G	53.36	0.125G	51.47	1.73G (2^{15})	96.95G	99.99	86.83

Table 5: 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. mem: memory used for the model. Note: the memory footprint in HLF is essentially independent of the number of classes used.

	HLF		**KNN**			**Kmeans**		
	2^{28}	2^{24}	S= 3%	6%	9%	S= 3%	6%	9%
L2	69.88	69.29	50.23	52.59	53.81	42.29	42.96	42.76
L3	47.90	46.64	30.93	32.67	33.71	31.63	31.56	31.53

Table 6: Accuracy comparison of hashing, KNN and Kmeans. HLF: joint hashing of labels and features. KNN: apply K Nearest Neighbor on sampled training set as search set. Kmeans: apply Kmeans on sampled training set to do clustering and then take its majority class as predicted class. S is the sample size which is the percentage of the entire training set.

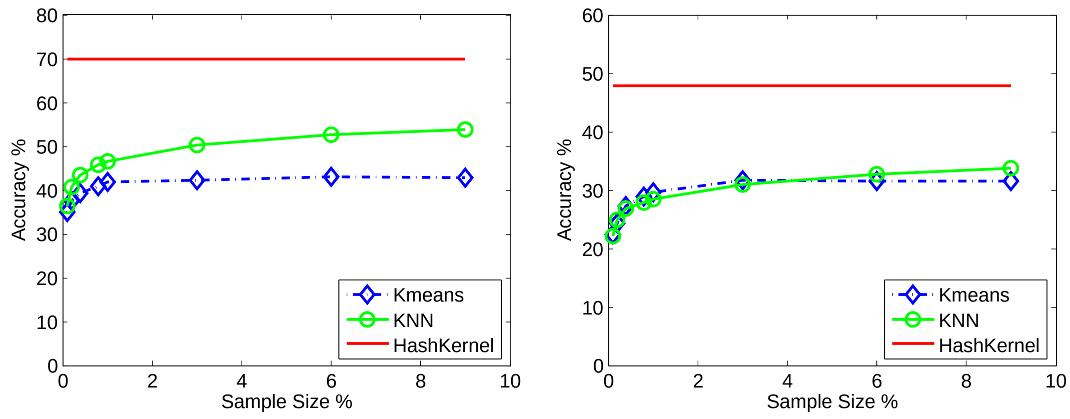


Figure 2: Test accuracy comparison of KNN and Kmeans on Dmoz with various sample sizes. Left: results on L2. Right: results on L3. Hash kernel (2^{28}) result is used as an upper bound.

Next we compare hash kernel with K Nearest Neighbor (KNN) and Kmeans. Running the naive KNN on the entire training set is very slow.[3]. Hence we introduce sampling to KNN. We first sample a subset from the entire training set as search set and then do KNN

3. In fact the complexity of KNN is $O(N \times T)$, where N, T are the size of the training set and the testing set. We estimate the running time for the original KNN, in a batch processing manner ignoring the data loading time, is roughly 44 days on a PC with a 3.2GHz cpu.

Data	Algorithm	Dim	Pre	TrainTest	Error %
L2	RP	2^7	779.98s	1258.12s	82.06%
	RP	2^8	1496.22s	3121.66s	72.66%
	RP	2^9	2914.85s	8734.25s	62.75%
	HK	2^7	0s	165.13s	62.28%
	HK	2^8	0s	165.63s	55.96%
	HK	2^9	0s	174.83s	50.98%
L3	RP	2^7	794.23s	18054.93s	89.46%
	RP	2^8	1483.71s	38613.51s	84.06%
	RP	2^9	2887.55s	163734.13s	77.25%
	HK	2^7	0s	1573.46s	76.31%
	HK	2^8	0s	1726.67s	71.93%
	HK	2^9	0s	1812.98s	67.18%

Table 7: Hash kernel vs. random projections with various feature dimensionalities on Dmoz. RP: random projections in Achlioptas (2003). HK: hash kernel. Dim: dimension of the new features. Pre: preprocessing time — generation of the random projected features. TrainTest: time to load data, train and test the model. Error: misclassification rate. Note that the TrainTest time for random projections increases as the new feature dimension increases, whereas for hash kernel the TrainTest is almost independent of the feature dimensionality. Moving the dimension from 2^8 to 2^9 the increasing in processing time of RP is not linear — we suspect this is because with 2^8 the RP model has $256 \times 7100 \times 8 \approx 14$MB, which is small enough to fit in the CPU cache (we are using a 4-cores cpu with a total cache size of 16MB), while with 2^9 the model has nearly 28MB, no longer fitting in the cache.

classification. To match the scheme of KNN, we use sampling in Kmeans too. Again we sample from the entire training set to do clustering. The number of clusters is the minimal number of classes which have at least 90% of the documents. Each test example is assigned to one of the clusters, and we take the majority class of the cluster as the predicted label of the test example. The accuracy plot in Figure 2 shows that in both Dmoz L2 and L3, KNN and Kmeans with various sample sizes get test accuracies of 30% to 20% off than the upper bound accuracy achieved by hash kernel. The trend of the KNN and Kmeans accuracy curve suggests that the bigger the sample size is, the less accuracy increment can be achieved by increasing it. A numerical result with selected sample sizes is reported in Table 6.

We also compare hash kernel with RP with various feature dimensionality on Dmoz. Here RP generates the random projected feature first and then does online learning and testing. It uses the same 4-cores implementation as hash kernel does. The numerical result with selected dimensionalities is in Table 7. It can be seen that hash kernel is not only much faster but also has much smaller error than RP given the same feature dimension. Note that both hash kernel and RP reduce the error as they increase the feature dimension. However, RP can't achieve competitive error to what hash kernel has in Table 5, simply because with large feature dimension RP is too slow — the estimated run time for RP with dimension 2^{19} on dmoz L3 is 2000 days.

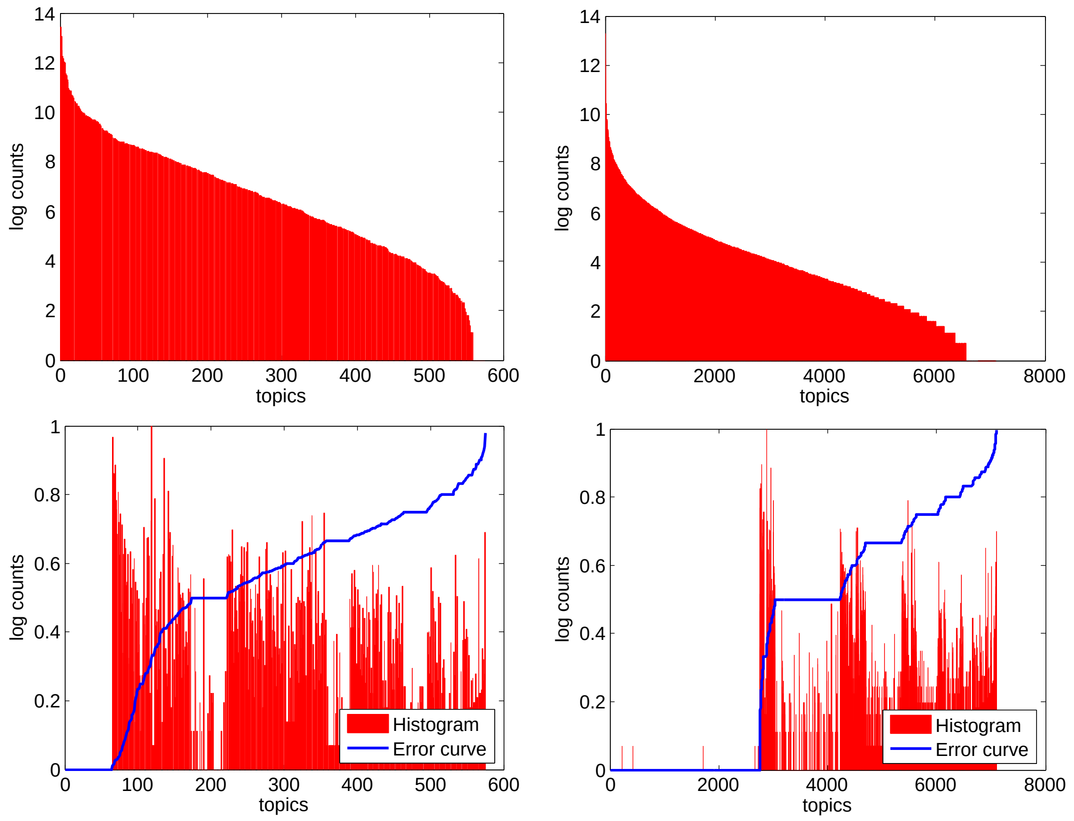


Figure 3: Left: results on L2. Right: results on L3. Top: frequency counts for topics as reported on the training set (the test set distribution is virtually identical). We see an exponential decay in counts. Bottom: log-counts and error probabilities on the test set. Note that the error is reasonably evenly distributed among the size of the classes (besides a number of near empty classes which are learned perfectly).

Furthermore we investigate whether such a 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 3 shows that 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".

6.3 Biochemistry and Bioinformatics Graph Classification

For the final experiment we work with graphs. The benchmark datasets we used here contain three real-world datasets: two molecular compounds datasets, (Debnath et al., 1991) and PTC (Toivonen et al., 2003), and a dataset for protein function prediction task (DD) from Dobson and Doig (2003). In this work we used the unlabeled version of these graphs, see e.g. Borgwardt et al. (2007).

All these datasets are made of sparse graphs. To capture the structure of the graphs, we sampled connected subgraphs with varying number of nodes, from $n = 4$ to $n = 9$.

Datasets	RW	SP	GKS	GK	HK	HKF
MUTAG	0.719	0.813	0.819	0.822	0.855	0.865
PTC	0.554	0.554	0.594	0.597	0.606	0.635
DD	>24h	>24h	0.745	>24h	0.799	0.841

Table 8: Classification accuracy on graph benchmark datasets. RW: random walk kernel, SP: shortest path kernel, GKS = graphlet kernel sampling 8497 graphlets, GK: graphlet kernel enumerating all graphlets exhaustively, HK: hash kernel, HKF: hash kernel with feature selection. '>24h' means computation did not finish within 24 hours.

Feature	All		Selection	
STATS	ACC	AUC	ACC	AUC
MUTAG	0.855	0.93	0.865	0.912
PTC	0.606	0.627	0.635	0.670
DD	0.799	0.81	0.841	0.918

Table 9: Non feature selection vs feature selection for hash kernel. All: all features. Selection: feature selection; ACC: accuracy; AUC: Area under ROC.

We used graph isomorphism techniques, implemented in Nauty McKay (1984) for getting a canonically-labeled isomorph of each sampled subgraph. The feature vector of each example (graph) is composed of the number of times each canonical isomorph was sampled. Each graph was sampled 10000 times for each of $n = 4, 5 \dots 9$. Note that the number of connected unlabeled graphs grows exponentially fast with the number of nodes, so the sampling is extremely sparse for large values of n. For this reason we normalized the counts so that for each dataset each feature of $\phi(x)$ satisfies $1 \geq \phi(x) \geq 0$.

We compare the proposed hash kernel (with/without feature selection) with random walk kernel, shortest path kernel and graphlet kernel on the benchmark datasets. From Table 8 we can see that the hash Kernel even without feature selection still significantly outperforms the other three kernels in terms of classification accuracy over all three benchmark datasets.

The dimensionality of the canonical isomorph representation is quite high and many features are extremely sparse, a feature selection step was taken that removed features suspected as non-informative. To this end, each feature was scored by the absolute vale of its correlation with the target. Only features with scores above median were retained. As can be seen in Table 9 feature selection on hash kernel can furthermore improve the test accuracy and area under ROC.

7. 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.

References

D. Achlioptas. Database-friendly random projections:johnson-lindenstrauss with binary coins. *Journal of Computer and System Sciences*, 66:671C687, June 2003.

Y. Altun and A.J. Smola. Unifying divergence minimization and statistical inference via convex duality. In H.U. Simon and G. Lugosi, editors, *Proc. Annual Conf. Computational Learning Theory*, LNCS, pages 139–153. Springer, 2006.

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

B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13:422C426, July 1970.

K. M. Borgwardt, H.-P. Kriegel, S. V. N. Vishwanathan, and N. Schraudolph. Graph kernels for disease outcome prediction from protein-protein interaction networks. In Russ B. Altman, A. Keith Dunker, Lawrence Hunter, Tiffany Murray, and Teri E Klein, editors, *Proceedings of the Pacific Symposium of Biocomputing 2007*, Maui Hawaii, January 2007. World Scientific.

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.

A. K. Debnath, R. L. Lopez de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *J Med Chem*, 34: 786–797, 1991.

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.

P. D. Dobson and A. J. Doig. Distinguishing enzyme structures from non-enzymes without alignments. *J Mol Biol*, 330(4):771–783, Jul 2003.

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* 30th *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.D. Lewis, Y. Yang, T.G. 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.W. Church, and T.J. 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.

B. D. McKay. nauty user's guide. Technical report, Dept. Computer Science, Austral. Nat. Univ., 1984.

A. Nobel and A. Dembo. A note on uniform laws of averages for dependent processes. *Statistics and Probability Letters*, 17:169–172, 1993.

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.C. 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.

Q. Shi, J. Petterson, G. Dror, J. Langford, A. Smola, A. Strehl, and S. V. N. Vishwanathan. Hash kernels. In Max Welling and David van Dyk, editors, *Proc. Intl. Workshop on Artificial Intelligence and Statistics*. Society for Artificial Intelligence and Statistics, 2009.

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.

H. Toivonen, A. Srinivasan, R. D. King, S. Kramer, and C. Helma. Statistical evaluation of the predictive toxicology challenge 2000-2001. *Bioinformatics*, 19(10):1183–1193, July 2003.

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

S. V. N. Vishwanathan, Karsten Borgwardt, and Nicol N. Schraudolph. Fast computation of graph kernels. In B. Schölkopf, J. Platt, and T. Hofmann, editors, *Advances in Neural Information Processing Systems 19*, Cambridge MA, 2007a. MIT Press.

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, 2007b.

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.