Nearest Neighbor Classification Using Bottom-k Sketches

Søren Dahlgaard*, Christian Igel*, Mikkel Thorup*†
*Department of Computer Science, University of Copenhagen
†AT&T Labs

***Abstract*—Bottom-k sketches are an alternative to $k\times$minwise sketches when using hashing to estimate the similarity of documents represented by shingles (or set similarity in general) in large-scale machine learning. They are faster to compute and have nicer theoretical properties. In the case of $k\times$minwise hashing, the bias introduced by not truly random hash function is independent of the number k of hashes, while this bias decreases with increasing k when employing bottom-k. In practice, bottom-k sketches can expedite classification systems if the trained classifiers are applied to many data points with a lot of features (i.e., to many documents encoded by a large number of shingles on average). An advantage of b-bit $k\times$minwise hashing is that it can be efficiently incorporated into machine learning methods relying on scalar products, such as support vector machines (SVMs). Still, experimental results indicate that a nearest neighbors classifier with bottom-k sketches can be preferable to using a linear SVM and b-bit $k\times$minwise hashing if the amount of *training* data is low or the number of features is high.**

***Keywords*-large-scale machine learning; hashing; nearest neighbor classification; set similarity; document encoding**

I. Introduction

Hashing algorithms have been successfully applied to scaling up the computation of set and bit-string similarities in large-scale machine learning and data mining (e.g., see [1], [2], [3], [4], [5]). The most prominent example is the comparison of text documents represented by bags of n-grams, where each document is represented by the set of all n-grams (i.e., sequences of n contiguous words) it contains, and the similarity of documents is computed as the similarity of the corresponding sets. Using a standard corpus (i.e., set of words that are considered), the sets get very large even for documents of moderate length and moderate value of n. Given the ever increasing digital document sources – the Internet being the most extreme example – it is obvious that efficient large-scale algorithms for computing or approximating set similarities are required.

Hashing algorithms can be employed to compute a short summary, a sketch, of the document representation, which can then be processed efficiently. Li and König [5] propose to use sketches generated by b-bit $k\times$minwise hashing [6], [7] for large-scale classification of documents. In this short paper, we explore using bottom-k sketches instead [8]. These are faster to compute and, as has been shown recently, have more appealing theoretical properties. However, in contrast to b-bit $k\times$minwise sketches, they do not directly lead to a scalar product.

The next section will briefly review minwise hashing for estimating set similarities. Section III will discuss bottom-k sketches and their properties. In section IV, experiments, similar to those conducted by [5], will be presented in order to validate the computational properties of bottom-k sketches for document classification.

II. Set similarity using hashing

We consider the Jaccard similarity or Jaccard index for quantifying the similarity of sets. For two sets A and B containing elements from a universe of discourse U (e.g., n-grams), the Jaccard index is defined as

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \ .$$

When A and B are huge, calculating the Jaccard similarity can be very time consuming, especially if we want to calculate the similarity of (almost) all pairs of sets in a big collection. Hashing can be employed to create a small signature (a sketch) for each set, which can be used to estimate the Jaccard similarity between any two of them.

We consider hash functions $h : U \to \{0,\dots,m-1\}$, $m \in \mathbb{N}$, and define $h(A) = \{h(a) \,|\, a \in A\}$ for $A \subseteq U$. Let us for now assume truly random hash functions and let h be such a function. If for each set A only the smallest hash value $\min(h(A))$ is stored, this value can be used to estimate the Jaccard index, because it holds that

$$R = \Pr\{\min(h(A)) = \min(h(B))\} = J(A,B) \ .$$

However, the variance of this estimator is much too high for practical purposes. Therefore, Broder et al. [1], [2] have introduced a technique using k hash functions $h_1,\dots,h_k$. This method is known as $k\times$minwise hashing and gives an unbiased estimator

$$\hat{R} = \frac{1}{k}\sum_{i=1}^{k}[\min(h_i(A)) = \min(h_i(B))]$$

of the Jaccard Index $J(A,B)$. Here $[P]$ is the Iverson bracket notation defined as $[P] = 1$ when P is true and 0 otherwise. This estimator has a variance $\mathrm{Var}(\hat{R}) = \frac{1}{k}R(1-R)$ decreasing with k.

A variant of $k\times$minwise sketches was presented in [6], [7]. The authors suggest only using the b least significant

978-1-4799-1293-3/13/$31.00 ©2013 IEEE

bits of each hash value, which results in less storage requirements. They analyze the variance of this new estimator and show that, when estimating resemblances $R \geq 0.5$, a lot of storage can be saved. For instance, choosing $b = 1$ only requires an increase in k of a factor of 3 in order to achieve the same variance as $b = 64$ (the original $\hat{R}$). This gives a $64/3$-fold improvement if one is only interested in resemblances greater than 0.5.

III. Bottom-k sketches

We will now consider an alternative to $k\times$minwise sketches called bottom-k sketches. This method described in [8] uses just a single hash function and stores the k smallest hash values instead of only one. Let the function k-min return the k smallest elements of a set, and let $S_{h,k}(A) = \text{k-min}(h(A))$ denote the k smallest hash values of a set A according to some hash function h. We drop the indices and write $S(A)$ if h and k are clear from the context. Given a truly random hash function, the bottom-k sketch $S(A)$ of a set A represents k random samples without replacement. We can use this random sample to estimate the relative size of some subset $Y \subseteq A$ as $|S(Y) \cap S(A)|/k$. In order to estimate the Jaccard similarity we need the relative size of $A \cap B$ to $A \cup B$. Using bottom-k sketches we get the unbiased estimator of the Jaccard index

$$\begin{aligned}\bar{R} &= \frac{|S(A) \cap S(B) \cap S(A \cup B)|}{k} \\ &= \frac{|S(A) \cap S(B) \cap \text{k-min}(S(A) \cup S(B))|}{k} \ .\end{aligned}$$

By using Chernoff bounds, we can see that this estimator has an expected error of $O(1/\sqrt{k})$. In the following, we will contrast $k\times$minwise and bottom-k sketches.

A. Sketch creation

The main advantage of bottom-k sketches compared to $k\times$minwise sketches is that sketch creation is much faster. With $k\times$minwise sketches, k hash functions must be evaluated for each member of the input set A. This leads to a sketch creation time of $O(|A|\cdot k)$, where $|A|$ is the cardinality of the input set A.

With bottom-k sketches just the single hash function h must be evaluated per element of the input set A. Instead the bulk of the work is in maintaining the set of the k smallest hash values. This can be done using a priority queue (max heap), yielding a sketch creation time of $O(|A| \cdot \lg k)$.

In the large-scale learning applications, the sizes $|A|$ of the input sets can be huge, and therefore the differences in sketch creation time can become a significant advantage of bottom-k.

B. Bias and variance

Under the assumption of truly random hash functions without collisions, the two estimators of the Jaccard similarity using $k\times$minwise and bottom-k sketches, respectively, are unbiased and have a standard deviation bounded by $1/\sqrt{Rk}$, where R is the real Jaccard similarity of the two sets. However, in practice we do not employ truly random hash functions and the situation is different.

When practical hash functions are considered, $k\times$minwise sketches lead to biased estimates of the Jaccard index – and this bias is independent of k. Two concepts are helpful to analyze how the methods work with practical hash functions. First, let us introduce the notion of a $(1\pm\varepsilon)$-minwise family of hash functions:

Definition 1: A family of hash functions $\mathcal{H}$ is said to be $(1 \pm \varepsilon)$-minwise if the probability over $h \in \mathcal{H}$ that any element $x \in U$ has the smallest hash value is off by at most a factor of $1 \pm \varepsilon$.

Clearly a truly random hash function is 1-minwise and using a $(1 \pm \varepsilon)$-minwise hash family causes a bias in the estimator of the Jaccard index of $(1 \pm \varepsilon)R$. Furthermore, recall the notion of k-independent hashing:

Definition 2: A family $\mathcal{H}$ of hash functions is k-independent or k-universal if for any k distinct keys $(x_1, \ldots, x_k) \in U^k$ and any k hash values $(y_1, \ldots, y_k) \in \{0, 1, \ldots, m-1\}^k$ we have

$$\Pr_{h\in\mathcal{H}}\{h(x_1) = y_1 \wedge \ldots \wedge h(x_k) = y_k\} = \frac{1}{m^k} \ .$$

For $k\times$minwise hashing it has been shown that there will always be bias in the estimation of the Jaccard index using practical hash functions. This is captured by the following theorem from [9]:

*Theorem 3 (*Pătraşcu & Thorup [9]): $(1 \pm \varepsilon)$-minwise hashing requires $\Omega(\lg \frac{1}{\varepsilon})$-independent hashing. Additionally, for 2-independent hash functions, there exists specific inputs leading to $\Theta(\lg n)$ bias no matter the value of k.

Because bottom-k and $k\times$minwise sketches are the same for $k = 1$ we might expect the same to hold for bottom-k sketches. This is however not the case.

*Theorem 4 (*Thorup [10]): Using bottom-k sketches with 2-independent, the expected relative error of the Jaccard index estimator is $O(1/\sqrt{Rk})$. In particular, the bias of the estimator vanishes as k increases.

This means that we can use the very fast multiply-shift hash function by Dietzfelbinger [11] together with bottom-k sketches and still have strong theoretic guarantees for the resemblance estimator.

The result can be viewed as a theoretical argument for using bottom-k sketches, however, we do not argue that the better theoretical guarantees matter in practice. Given enough randomness in the training data, loosely speaking, even simple hash functions work as well as a truly random hash function [12], and thus the bias vanishes in both approaches.

C. Minwise hashing and linear kernels

A big advantage of $k\times$minwise sketches over bottom-k sketches is that they give rise to a Mercer kernel

function. This implies, for example, that we can use the $k\times$minwise sketches in conjunction with support vector machines (SVMs, [13]).

The estimate of the Jaccard index by $k\times$minwise sketches is a Mercer kernel on the power set $\mathcal{P}(U)$ of U. Using it as a kernel in a nonlinear SVM, however, exhibited too slow training times according to [5]. Instead they suggest using b-bit $k\times$minwise sketches to create a linear kernel, which can be used with the very efficient linear SVM such as LIBLINEAR [14] on very large scale data sets.

*Theorem 5 (*Li, Shrivastava, Moore, & C. König [5]): Let us consider ℓ sets $A_1, \ldots, A_\ell \subset U$ and k hash functions $h_1, \ldots, h_k$. The $\ell \times \ell$ kernel (Gram) matrix $\vec{M}$ defined by b-bit minwise hashing with components

$$M_{ij} = \sum_{l=1}^{k} \left(\prod_{t=1}^{b} [\min(h_l(A_i))_t = \min(h_l(A_j))_t] \right) ,$$

where $\min(h(A_i))_t$ is the t-th least significant bit of $\min(h(A_i))$, is positive semidefinite.

This theorem also shows how to integrate b-bit $k\times$minwise hashing with the linear SVM. This is done by letting each minhash correspond to a vector of size 2^b, which is all 0s except having a 1 in the position of the b least significant bits (i.e., at the position indexed by the integer representation of the b bits) of the minhash. All these vectors are then concatenated to form a final vector of size $2^b k$ with exactly k 1s. As an example using $k = 3$ and $b = 2$ the $k\times$minwise sketch $(12, 34, 51)$ would be represented by the vector

$$(0, 0, 0, 1, \quad 0, 1, 0, 0, \quad 1, 0, 0, 0)^T ,$$

because the two lowest bits of 12, 34, and 51 are 00, 10, and 11, respectively. This approach was successfully used in [5] and serves the baseline for our comparison in the next section.

IV. Experiments

This section will investigate whether the differences between the two hashing schemes matter in practice. First, we will validate that the computation of bottom-k sketches is significantly faster compared to $k\times$minwise sketches already on a rather small data set. Second, we will compare the two approaches in classification experiments.

In our experiments, we compared the performance of a nearest neighbor (NN) classifier using bottom-k sketches with the approach described in [5], which utilizes the b-bit $k\times$minwise sketch together with a linear soft-margin SVM. From the theoretical discussion in section III, it is clear that bottom-k sketches always perform better (timewise) than $k\times$minwise sketches if we can directly interchange the two. Therefore, we did not perform experiments using $k\times$minwise sketches with an NN classifier. However, we measured the sketch creation times to verify that such a comparison would indeed be trivial.

The focus of this study is classification time, and the goal is to increase speed for certain scenarios without a deterioration of accuracy. The classification time of the NN method scales linearly with the number of training data points. In contrast, the number of training data points does not affect the *linear* SVM. After the representation of the input is computed as described in subsection III-C, the SVM basically just needs to calculate the scalar product between sparse $k2^b$-dimensional binary vectors. Thus, it is clear that the b-bit $k\times$minwise linear SVM will always be faster than the bottom-k NN if the size of the training set increases beyond a certain threshold. However, we wanted to study if the K-NN with bottom-k outperforms the b-bit $k\times$minwise SVM in relevant scenarios in which the training set is small and the resulting classifier is applied to many large input sets. This scenario occurs, for example, if a document database is large and labels are expensive to obtain.

A. Data sets

We wanted to stay close to the experimental evaluation in [5] and therefore considered the WEBSPAM binary classification task. The data set has $350,000$ elements and is available from the LibSVM homepage [15]. Each input describes a web page by a set of 3-grams (we do not take into account if a 3-gram occurs several times). Each 3-gram is also referred to as a feature. The total number of different features is $D = 16,609,143$. Thus, each input corresponds to a sparse D-dimensional binary vector. The average number features (i.e., the average set size) is as low as $3,727$.

In addition, we also tested on the NEWS20 binary classification task as described in [16]. The data set, which is much smaller than WEBSPAM, is provided on the LibSVM homepage [15] and consists of 3-gram representations of postings to newsgroups. It only contains $19,996$ elements with a total number of $D = 1,355,191$ different features. The average number of features is only 454.

In our experiments comparing classifiers, we vary the training data set size. In contrast to other studies, we consider rather small training data set sizes. For large training data sets, the classification time of exact NN, which scales linearly with the number of training data points, will clearly be worse compared to the linear SVM. The number of training points was selected based on the average number of features of the data set. Because the $k\times$minwise sketch creation is linear in the amount of features, we expect that average number of features roughly determines the threshold below which the NN classifier outperforms the linear SVM in terms of time. The training set was selected at random. We considered 750, 1000, and 1250 documents for training in the case of the WEBSPAM data set, and 100, 150, and 200 documents in the case of the NEWS20 data set. The remaining data points were used for testing. For linear SVM experiments with larger training set sizes we refer to [5].

Because the experiments were randomized (hash functions as well as training points), each experiment was repeated 10 times.

B. Hash functions

The hash function used in all experiments was Dietzfelbinger's efficient multiply-shift hash function. He proves that for the family of hash functions defined by

$$h_{a,b}(x) = \left\lfloor \frac{((ax+b) \bmod km)}{k} \right\rfloor ,$$

where $k > 1$ is a power of 2 and $a, b \in \{0, 1, \ldots, km-1\}$, is 2-independent [11].

The usefulness of this family stems from the implementation details. Because k is power of 2, we can implement the division by a shift operation. Also, if we assume 32-bit integers, we can pick $km = 2^{32}$ and avoid an explicit modulus operation. A C-implementation could look like

$$(x \ * \ a \ + \ b) \ >> \ \lg(k)$$

and requires only few clock cycles.

C. Experimental setup

We used LIBLINEAR [14] for the linear SVM and a simple K-NN implementation in C++. All tests were run on a computer with Intel(R) Core(TM) i7 CPU @ 2.13GHz with 4GB memory running Archlinux 3.4.4-3. Thus, memory was sparse, demonstrating the power of using sketches as, for instance, the original WEBSPAM data set requires 24GB of space.

As in [5], the testing time measurements also included the data loading, as designed by LIBLINEAR. This, however, can be neglected compared to the sketch creation time. Contrary to the times reported in [5], we also included the time it took to create the sketches for the input. This is the majority of the work and should not be neglected.

D. Model selection

Both the linear SVM and the K-NN algorithm have a hyperparameter, C and K, respectively. Li and König [5] performed an extensive comparison of various C values, which helped us to narrow down the possible value for the WEBSPAM experiments. We tested $C \in \{0.1, 1, 10\}$. In accordance with [5], $C = 1$ gave the best results in all our experiments. Therefore, we only report results for $C = 1$ in the following. We did not tune K and set it to $K = 1$. In the case of the NEWS20 data set, we selected the regularization parameter C from $\{10^{-4}, 10^{-3}, \ldots, 100\}$ using 5-fold cross-validation on the training data [17].

For k we used the typical values $k = 100$ and $k = 200$ which worked well in the experiments in [5] and in previous studies [2], [1], [6]. For b we only used the value 3.

E. Results

Table I shows how long it took to compute the two types of sketches for the two data sets depending on k. Even these results on rather small data sets clearly show that the sketch creation for bottom-k is much faster than for $k\times$minwise. For example, the average parsing time of WEBSPAM with bottom-k sketches for $k = 200$ was around 70 seconds, while it was around 468 seconds for $k\times$minwise. Such a significant improvement would also be observed when comparing NN classifiers using the different types of sketches. As argued before, in this case our approach is clearly superior.

Data set	WEBSPAM		NEWS20	
Sketch	Bot-k	$k\times$min	Bot-k	$k\times$min
$k = 100$	60.56s	240.16s	0.44s	1.85s
$k = 200$	70.12s	468.53s	0.49s	3.59s

Table I
AVERAGE PARSING TIME FOR BOTTOM-k AND $k\times$MINWISE ON THE $350,000$ ELEMENTS OF THE WEBSPAM DATA SET AND ON THE $19,996$ ELEMENTS OF THE NEWS20 DATA SET. THE RESULTS ARE AVERAGED OVER 30 TRIALS.

We use box plots displaying the median as well as the lower and upper quartile to visualize the results of our classification experiments. Figure 1 provides the accuracies for the WEBSPAM data set with $k = 100$ and $k = 200$, respectively. The results were very close to those reported in [5] even though we only used a fraction of the available data for training – down to less than 1% of the available data. The nearest neighbor classifier was consistently better than the SVM for $k = 100$, for $k = 200$ the two methods perform on par.

The classification accuracies show that employing 1-NN is a reasonable approach to solving the considered benchmark tasks. However, one has to keep in mind that using too few bits in the b-bit $k\times$minwise sketches may deteriorate the classification performance, in particular if k is too small. Thus, increasing b may improve the SVM accuracies, but would also slow down the method – and in this study our main concern is the comparison of classification times.

Figure 2 provides the testing time for the WEBSPAM data set using $k = 100$ and $k = 200$ respectively. As expected, there was a clear linear relation between the number of training points and the testing times for K-NN.

As hypothesized, K-NN with bottom-k was faster than the SVM relying on b-bit $k\times$minwise sketches for small training data set sizes. The two methods were close to each other at 1000 training points, which is about $1/4$ of the average feature count. For larger training data set sizes, the linear SVM classifier was faster than K-NN.

Classifying the NEWS20 yielded similar results. Here the linear SVM had problems learning a good hypotheses given

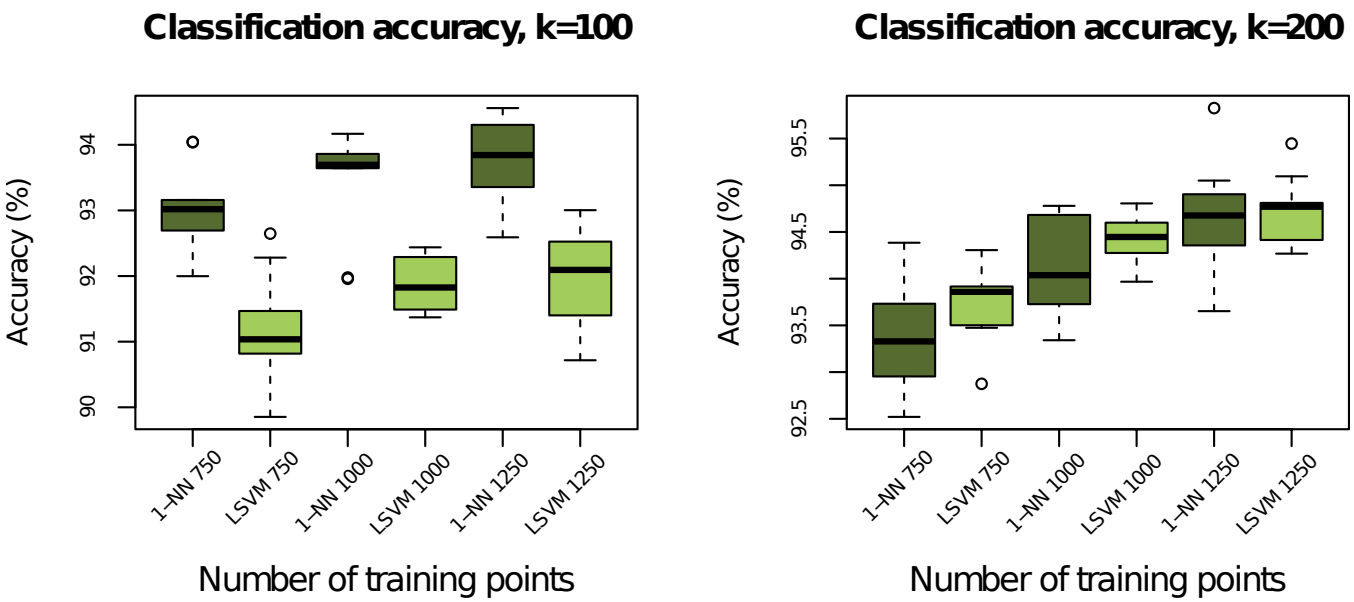

Figure 1. Classification accuracies for the WEBSPAM data set using $k = 100$ and $k = 200$. The results for the SVM are for $C = 1$, which gave the best results. The results are based on 10 repetitions. The dark boxes are for the 1-NN classifier and the light boxes are for the linear SVM.

the very few training samples (and $b = 3$). The NN classifier performed significantly better, see Figure 3.

Figure 4 again shows the expected linear relation in the NN testing times. The NN with bottom-k became faster than the SVM when the training data set was reduced to approx. 150 points, which is about $1/3$ of the average feature count.

V. Conclusions

This study considered bottom-k sketches as an alternative to $k\times$minwise sketches when using hashing to estimate the Jaccard similarity of sets in large-scale machine learning. Bottom-k sketches

- require only $O(|A| \cdot \lg k)$ computations for a set of size $|A|$ in contrast to $O(|A| \cdot k)$ computations needed by $k\times$minwise sketches;
- provide an estimate of the Jaccard similarity with bias and variance decreasing with the number of hashes k if a 2-independent hash function is used, while $k\times$minwise sketches lead to a biased estimate for any hash function with constant independence independent of k.

While the latter point is mainly of theoretical interest, we experimentally showed that bottom-k hashing can indeed significantly speed-up classification algorithms in practice. In particular, this is the case for applications where

- the classifier is applied to many data points with a lot of features (i.e., large sets),
- a learning algorithm just requiring a metric space (and no vector space) is a proper choice.

An example would be classifying a huge amount of web pages using a nearest neighbor classifier trained with only a few hand-labeled samples.

However, b-bit $k\times$minwise hashing can be used to create a fixed-length vector representation of the input sets that can directly be used with a linear kernel function, which allows using highly efficient linear classification methods. It is left to future work to develop a similar representation based on bottom-k hashing or to prove that this is not possible.

Acknowledgment

Søren Dahlgaard and Christian Igel gratefully acknowledge support from the European Commission through project AKMI (PCIG10-GA-2011-303655).

References

[1] A. Z. Broder, "On the resemblance and containment of documents," in *Compression and Complexity of Sequences*. IEEE Press, 1997, pp. 21–29.

[2] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig, "Syntactic clustering of the web," *Computer Networks*, vol. 29, no. 8-13, pp. 1157–1166, 1997.

[3] K. Q. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Altenberg, "Feature hashing for large scale multitask learning," in *Proceedings of the 26th International Conference on Machine Learning (ICML)*. ACM, 2009, pp. 1113–1120.

[4] Q. Shi, J. Petterson, G. Dror, J. Langford, A. Smola, and S. Vishwanathan, "Hash kernels for structured data," *Journal of Machine Learning Research*, vol. 10, pp. 2615–2637, 2009.

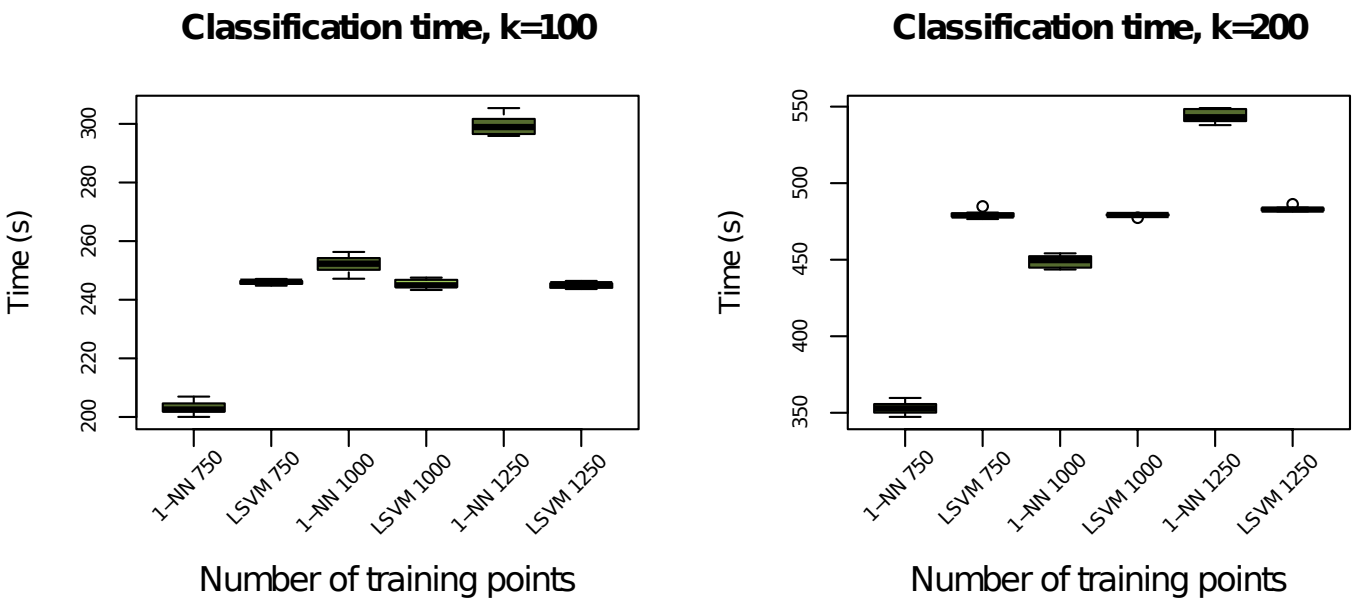

Figure 2. Testing times on the WEBSPAM data set using $k = 100$ and $k = 200$. The results for the SVM are for $C = 1$, which gave the best results. The results are base on 10 repetitions. The dark boxes are for the 1-NN classifier and the light boxes are for the linear SVM.

[5] P. Li, A. Shrivastava, J. L. Moore, and A. C. König, "Hashing algorithms for large-scale learning," in *Advances in Neural Information Processing Systems 24 (NIPS)*, J. Shawe-Taylor, R. S. Zemel, P. Bartlett, F. C. N. Pereira, and K. Q. Weinberger, Eds., 2011, pp. 2672–2680.

[6] P. Li and A. C. König, "b-bit minwise hashing," in *Nineteenth International World Wide Web Conference (WWW 2010)*, M. Rappa, P. Jones, J. Freire, and S. Chakrabarti, Eds. ACM, 2010, p. 671680.

[7] ——, "Theory and applications of b-bit minwise hashing," *Communications of the ACM*, vol. 54, no. 8, pp. 101–109, 2011.

[8] E. Cohen and H. Kaplan, "Summarizing data using bottom-k sketches," in *Proceedings of the Twenty-sixth Annual ACM Symposium on Principles of Distributed Computing (PODC)*. ACM, 2007, pp. 225–234.

[9] M. Pătraşcu and M. Thorup, "On the k-independence required by linear probing and minwise independence," in *Proceedings of the 37th International Colloquium on Automata, Languages and Programming (ICALP)*. Springer-Verlag, 2010, pp. 715–726.

[10] M. Thorup, "Bottom-k and priority sampling, set similarity and subset sums with minimal independence," in *ACM Symposium on the Theory of Computing (STOC)*. ACM, 2013, pp. 371–380.

[11] M. Dietzfelbinger, "Universal hashing and k-wise independent random variables via integer arithmetic without primes," in *13th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, ser. LNCS, C. Puech and R. Reischuk, Eds., vol. 1046. Springer, 1996, pp. 569–580.

[12] M. Mitzenmacher and S. P. Vadhan, "Why simple hash functions work: exploiting the entropy in a data stream," in *Proceedings of the 19th annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2008, pp. 746–755.

[13] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[14] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "LIBLINEAR: A library for large linear classification," *Journal of Machine Learning Research*, vol. 9, pp. 1871–1874, 2008.

[15] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, 2011.

[16] S. Keerthi and D. DeCoste, "A modified finite Newton method for fast solution of large scale linear SVMs," *Journal of Machine Learning Research*, vol. 6, pp. 341–361, 2005.

[17] T. Hastie, R. Tibshirani, and J. Friedmann, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed. Springer-Verlag, 2009.

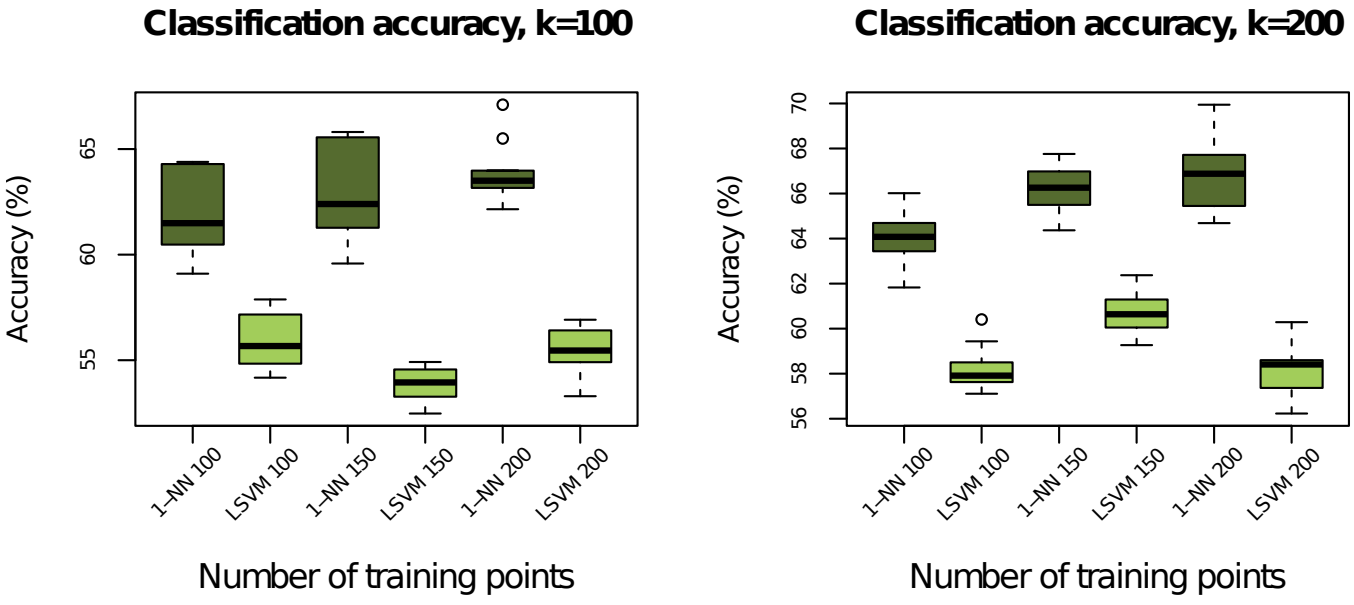

Figure 3. Classification accuracies on the NEWS20 data set for $k = 100$ and $k = 200$. The results are base on 10 repetitions. The dark boxes are for the 1-NN classifier and the light boxes are for the linear SVM.

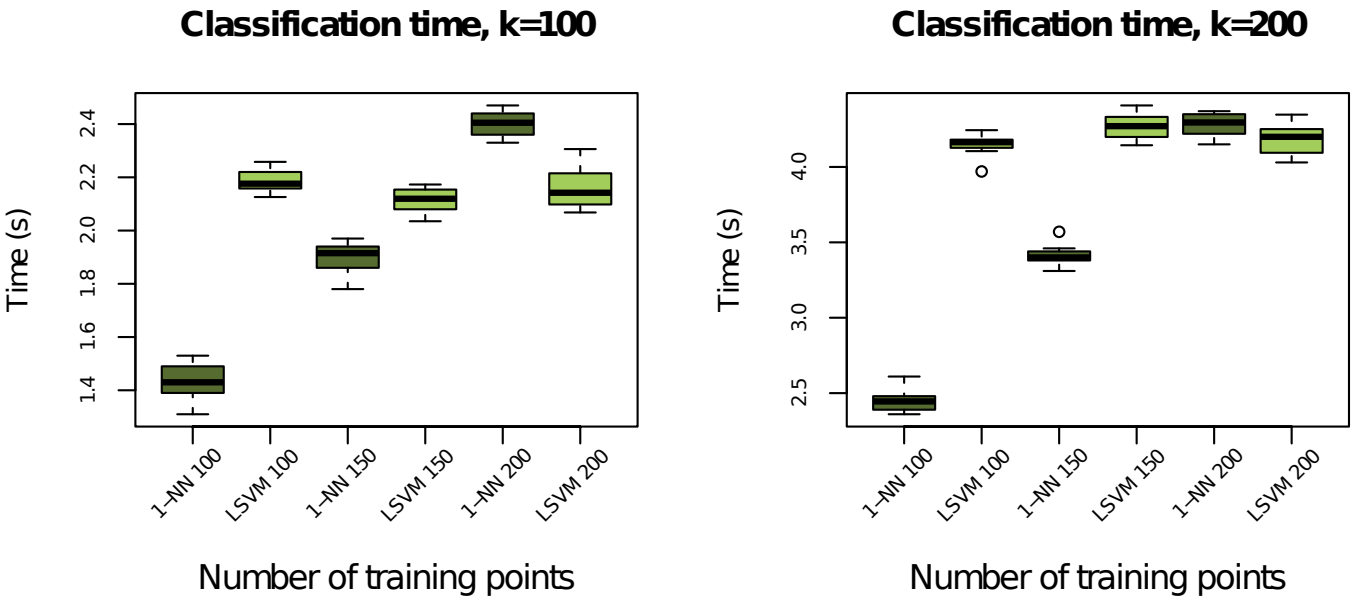

Figure 4. Testing times on the NEWS20 data set for $k = 100$ and $k = 200$. The results are base on 10 repetitions. The dark boxes are for the 1-NN classifier and the light boxes are for the linear SVM.