# Combining Hashing and Abstraction in Sparse High Dimensional Feature Spaces

**Cornelia Caragea**
The Pennsylvania State University
College Park, PA 16801, USA
ccaragea@ist.psu.edu

**Adrian Silvescu**
Naviance Inc.
Washington, D.C., USA
silvescu@gmail.com

**Prasenjit Mitra**
The Pennsylvania State University
College Park, PA 16801, USA
pmitra@ist.psu.edu

**Abstract**

With the exponential increase in the number of documents available online, e.g., news articles, weblogs, scientific documents, the development of *effective* and *efficient* classification methods is needed. The performance of document classifiers critically depends, among other things, on the choice of the feature representation. The commonly used "bag of words" and $n$-gram representations can result in prohibitively high dimensional input spaces. Data mining algorithms applied to these input spaces may be *intractable* due to the large number of dimensions. Thus, dimensionality reduction algorithms that can process data into features fast at runtime, ideally in *constant* time per feature, are greatly needed in high throughput applications, where the number of features and data points can be in the order of millions. One promising line of research to dimensionality reduction is feature clustering. We propose to combine two types of feature clustering, namely hashing and abstraction based on hierarchical agglomerative clustering, in order to take advantage of the strengths of both techniques. Experimental results on two text data sets show that the combined approach uses significantly smaller number of features and gives similar performance when compared with the "bag of words" and $n$-gram approaches.

## Introduction

Recent World Wide Web advances have resulted in large amounts of online text data such as news articles, blogs, and scientific documents. The proliferation of such data poses several challenges to the data mining community. In particular, these data require *effective* and *efficient* methods for text classification, ranking, organization, indexing, and summarization. The "bag of words" and $n$-gram feature representations, commonly used for text classification, usually result in *prohibitively* high dimensional input spaces. Data mining algorithms applied to these input spaces may be intractable due to the large number of dimensions. Hence, using dimensionality reduction techniques can be crucial for the performance and the complexity of the learning algorithms.

The main idea behind dimensionality reduction is to project the original high-dimensional data into a lower-dimensional space, in which patterns in the data can be more easily identified. Models such as Principal Component Analysis (PCA) (Jolliffe 1986), Latent Dirichlet Allocation (LDA) (Blei, Ng, and Jordan 2003), Probabilistic Latent Semantic Analysis (PLSA) (Hofmann 1999), Latent Semantic Indexing (LSI) (Deerwester et al. 1990) are widely used to perform dimensionality reduction for text classification.

Unfortunately, for very high-dimensional data, with hundreds of thousands of dimensions, processing data instances into feature vectors at runtime, using these models, is computationally very expensive. For example, given a *learned* LDA model $\mathcal{M}$ and a new instance $\mathbf{x}$, representing $\mathbf{x}$ as a feature vector requires inference of unknown hidden topics $\theta$, i.e., the posterior probability $p(\theta|\mathbf{x}, \mathcal{M})$ needs to be estimated. Because the number of new instances can be in the order of millions in high throughput data mining applications such as *online* text classification, ranking, ad selection, these approaches are not necessarily very efficient. Therefore, dimensionality reduction algorithms that can process data into features fast at runtime, ideally in *constant* time per feature, are greatly needed.

One promising line of research to dimensionality reduction is feature clustering. Two types of feature clustering are by bottom-up hierarchical agglomerative clustering (Duda, Hart, and Stork 2001) (e.g., using the Jensen-Shannon divergence (Lin 1991)), and by hashing (or random clustering).

Feature clustering by hierarchical agglomerative clustering finds clusters of "similar" features, called *abstractions*, where the feature similarity is measured by distance metrics such as the Jensen-Shannon divergence between the probability distributions of the class variable given the features (Baker and McCallum 1998; Silvescu, Caragea, and Honavar 2009; Slonim and Tishby 1999). High-dimensional feature vectors are then compressed by *abstracting out* the differences between features in the same cluster and adding up their counts. We will refer to this as feature abstraction. Feature abstraction effectively reduces the number of features by one to three orders of magnitude without sacrificing classification performance (Baker and McCallum 1998; Silvescu, Caragea, and Honavar 2009), while maintaing a constant processing time per feature at runtime (i.e., evaluating abstractions needs only an array entry look-up). However, one of the main shortcomings of feature abstraction is that, for very high dimensional spaces, e.g., $d = 2^{26}$, it requires $O(d)$ space to store the array that specifies the mem-

bership of features into abstractions. For very large values of $d$, storing the array and performing array entry look-up operations may become difficult.

Feature hashing (Shi et al. 2009; Weinberger et al. 2009; Forman and Kirshenbaum 2008; Langford, Li, and Strehl 2007) offers a very inexpensive, yet effective, approach to reducing the number of features. Feature hashing allows random collisions into the latent factors. The original high-dimensional space is "reduced" by *hashing* the features, using a hash function, into a lower-dimensional space, i.e., mapping features to hash keys, where multiple features can be mapped (at random) to the same key, and "aggregating" their counts. Although this method is very effective in reducing the number of features from very high (e.g., $2^{26}$) to mid-size (e.g., $2^{16}$) dimensions, feature hashing can result in significant loss of information, especially when hash collisions occur between highly frequent features, which can have significantly different class distributions.

Given the complementary strengths of hashing and abstraction, one question that can be raised is the following: *Can one design an effective and efficient approach to reduce the number of feature dimensions by exploiting the strengths of these techniques (hashing and abstraction) and overcome their limitations?* The research that we describe in this paper addresses specifically this question.

**Contributions.** We present an approach to dimensionality reduction that combines hashing and abstraction to generate accurate and concise models, while maintaing a constant processing time per feature at runtime. More precisely, we propose to first hash the original high-dimensional spaces to mid-size dimensions (e.g., $2^{16}$ or $2^{14}$), operation which does not significantly distort the data, and then use abstraction to further reduce the dimensionality for a small or no loss in performance. The use of hashing minimizes the space requirement needed by abstraction. The use of abstraction enforces collisions between features that have "similar" class distributions, as opposed to allowing random collisions, which could result in significant loss of information.

We empirically show that combining hashing and abstraction: (a) can result in models that use a significantly smaller number of features, and have similar performance, or sometimes better, compared to the "bag of words" approach; (b) can provide a way to handle very high dimensionality that results from using $n$-grams (i.e., sequences of $n$ contiguous words); (c) significantly outperforms the combination of hashing and feature selection (Guyon and Elisseeff 2003).

## Related Work

A variety of approaches to dimensionality reduction have been studied in the literature as described below.

*Feature selection* (Guyon and Elisseeff 2003; Yang and Pederson 1997) reduces the number of features by selecting a subset of the available features based on some chosen criteria. For example, feature selection by average mutual information selects the top words that have the highest average mutual information with the class variable (Yang and Pederson 1997).

*Principal Component Analysis (PCA)* (Jolliffe 1986) finds a projection of the original $d$-dimensional input space into a lower $m$-dimensional orthogonal space, which captures as much of the data variance as possible, by exploiting the eigen-structure of the data covariance matrix. PCA is computationally very expensive (Golub and van Loan 1983), although less expensive methods for finding only $m$ eigenvectors and eigenvalues were developed (see for example (Roweis 1998)). *Singular Value Decomposition (SVD)* (Berry 1992; Deerwester et al. 1990) is directly applied to the data matrix. SVD is also computationally expensive, although SVD methods for sparse text data were developed with smaller computational complexity (Papadimitriou et al. 1998). However, neither PCA nor SVD is efficient to reduce the dimensionality for a large number of new instances.

*Topic models* such as Latent Dirichlet Allocation (LDA) (Blei, Ng, and Jordan 2003), Probabilistic Latent Semantic Analysis (PLSA) (Hofmann 1999), and Latent Semantic Indexing (LSI) (Deerwester et al. 1990) are dimensionality reduction models, designed to uncover hidden *topics*, i.e., clusters of semantically related words that co-occur in the documents. LSI uses SVD to identify topics, which are then used to represent documents in a low dimensional "topic" space. Hence, it is also inefficient. LDA models each document as a mixture of topics, and each topic as a distribution over the words in the vocabulary. The topic distribution of a document can be seen as a low-dimensional "topic" representation. However, LDA requires inference at runtime to estimate the topic distribution.

*Random projections* (Johnson and Lindenstrauss 1984; Achlioptas 2003; Liberty, Ailon, and Singer 2008; Bingham and Mannila 2001; Rahimi and Recht 2008) project high $d$-dimensional spaces into lower $m$-dimensional spaces using a random $m \times d$ matrix $R$ with unit length columns. Bingham and Mannila (2001) showed empirically that random projections do not significantly distort the data and, although their performance is not as accurate as that of SVD, they are computationally less complex than SVD.

*Feature hashing (or random clustering).* Shi et al. (2009) and Weinberger et al. (2009) presented hash kernels to map high dimensional input spaces into low dimensional spaces and showed them to be highly accurate for large scale classification and large scale multitask learning when the dimension of the low space is sufficiently large. Ganchev and Dredze (2008) empirically showed that hash features can produce accurate results on various NLP applications. Forman and Kirshenbaum (2008) proposed a fast feature extraction approach by combining parsing and hashing for text classification and indexing. Indyk and Motwani (1998) and Gionis, Indyk and Motwani (1999) presented Locality-Sensitive Hashing (LSH), a random hashing/projection technique for approximate nearest-neighbor query problems. Objects (e.g., images or text documents) are hashed using multiple hash functions such that, for each hash function, collisions are more likely to happen between objects that are close to each other, rather than objects that are far apart. A query object is then hashed, using the same hash functions, and the objects stored in buckets containing the query object are ranked and retrieved as the nearest neighbors. However, LSH is computationally more expensive than hash kernels, which require only adding up the vector coordinates with the

same hash key (Shi et al. 2009).

*Feature Abstraction.* Information Bottleneck (IB) and its variants (Tishby, Pereira, and Bialek 1999; Slonim and Tishby 1999; Pereira, Tishby, and Lee 1993) are distributional clustering based approaches, designed to find a compression of a variable $X$ while preserving as much information as possible about a target variable $Y$. Baker and McCallum (1998) applied distributional clustering to reduce the dimensionality of the feature space for document classification tasks. Silvescu et al. (2009) applied agglomerative IB (Slonim and Tishby 1999) to simplify the data representation on biological sequence classification tasks. Other feature abstraction methods include: automatic construction of word taxonomies from text data and their usage to text classification (Kang et al. 2005); compression of conditional probability tables in Bayesian networks using abstraction-based search (desJardins, Getoor, and Koller 2000).

In contrast to the approaches discussed above, we present a hybrid approach that combines hashing (Shi et al. 2009) and abstraction (Silvescu, Caragea, and Honavar 2009) to exploit their strengths, and address and minimize their limitations. As feature selection is yet another accurate and efficient dimensionality reduction method that can be combined with feature hashing, we compare our approach with the combination of hashing and feature selection.

## Methods

The "bag of words" and $n$-gram approaches construct a vocabulary of size $d$, which contains all words or $n$-grams in a collection of documents. A document is represented as a vector $\mathbf{x}$ with as many entries as the words or $n$-grams in the vocabulary, where an entry $k$ in $\mathbf{x}$ can record the frequency (in the document) of the $k^{th}$ word or $n$-gram in the vocabulary, denoted by $x_k$. Because only a small number of words (compared to the vocabulary size) occurs in a document, the representation of $\mathbf{x}$ is very sparse, i.e., only a small number of entries of $\mathbf{x}$ is non-zero. However, storing the parameter vectors in the original input space requires $O(d)$ numbers, which can become difficult given today's very large collections of documents. The combination of hashing and abstraction helps reduce the size of the parameter vectors.

### Feature Hashing

Feature hashing (Shi et al. 2009; Weinberger et al. 2009; Forman and Kirshenbaum 2008; Langford, Li, and Strehl 2007) is a dimensionality reduction technique, in which high-dimensional input vectors $\mathbf{x}$ of size $d$ are *hashed* into lower-dimensional feature vectors $\mathbf{x^h}$ of size $b$. Let $\mathcal{S}$ denote the set of all possible strings and $h$ and $\xi$ be two hash functions, such that $h : \mathcal{S} \rightarrow \{0, \cdots, b-1\}$ and $\xi : \mathcal{S} \rightarrow \{\pm 1\}$, respectively. Each token in a document is directly mapped, using $h$[1], into a hash key, which represents the index of the token in the feature vector $\mathbf{x^h}$, such that the hash key is a number between 0 and $b-1$. Each index in $\mathbf{x^h}$ stores the value ("frequency counts") of the corresponding hash feature. The hash function $\xi$ indicates whether to increment or decrement the hash dimension of the token, which renders the hash feature vector $\mathbf{x^h}$ to be unbiased (see (Weinberger et al. 2009) for more details).

Thus, an entry $i$ in $\mathbf{x^h}$ records the "frequency counts" of tokens that are hashed together, at random, into the same hash key $i$. That is, $x_i^h = \sum_{k:h(k)=i} \xi(k) x_k$, for $k = 0, \cdots, d-1$ and $i = 0, \cdots, b-1$. Note that in the case of $\xi \equiv 1$, $x_i^h$ represents the actual frequency counts.

As can be seen, multiple tokens can be mapped, through $h$, into the same hash key. According to the birthday paradox, if there are at least $\sqrt{b}$ features, then collisions are likely to happen (Shi et al. 2009), and hence, useful information necessary for high accuracy classification could be lost through feature hashing. However, words in a document collection typically follow a Zipf distribution, i.e., only very few words occur with high frequency, whereas the majority of them occur very rarely. Because hash collisions are independent of word frequencies, most collisions are likely to happen between infrequent words. Weinberger et al. (2009) have proven that, for a feature vector $\mathbf{x}$ such that $\|\mathbf{x}\|_2 = 1$, the length of $\mathbf{x}$ is preserved with high probability, for a sufficiently large dimension (or hash size) $b$ and a sufficiently small magnitude of $\mathbf{x}$, i.e., $\|\mathbf{x}\|_\infty$ (lower and upper bounds are theoretically derived (Weinberger et al. 2009)).

However, for many practical applications, the value of $b$ can be smaller than the theoretical lower bound. This may be problematic as the smaller the size $b$ of the hash vector $\mathbf{x^h}$ becomes, the more collisions occur in the data. Even a single collision of very high frequency words, with different class distributions, can result in significant loss of information. In order to avoid such "bad" collisions, we propose to combine feature hashing (Shi et al. 2009; Weinberger et al. 2009) with feature abstraction (Silvescu, Caragea, and Honavar 2009). Specifically, we first analyze what is the dimension $b$ at which the performance of classifiers that use hash features starts degrading due to "bad" hash collisions. We propose to use feature hashing to reduce very high dimensional input vectors (e.g., $2^{26}$) into mid-size dimensional hash vectors (e.g., $2^{16}$), before the performance starts degrading, and further reduce the dimensionality of the hash vectors to smaller dimensions (e.g., to $2^{10}$) using feature abstraction that groups together hash features with "similar" class distributions. Next, we present the combination of feature hashing with feature abstraction.

### Feature Abstraction over Hash Vectors

Feature abstraction effectively reduces a classifier input size by clustering *similar* features into an abstraction hierarchy.

Let $\mathcal{H}$ denote a set of hash features of size $b$, and $m$ denote the dimension of the desired reduced space for abstraction ($m \ll b$). An *abstraction hierarchy* (AH) $\mathcal{T}$ over $\mathcal{H}$ is defined as a rooted tree such that the leaf nodes correspond to the hash features $h_i$ in $\mathcal{H}$ and the internal nodes correspond to abstractions or clusters of hash features. An $m$-cut $\gamma_m$ through $\mathcal{T}$ is a set of $m$ nodes, which forms a partition of $\mathcal{H}$, that is, $\gamma_m = \{a_1 : \mathcal{H}_1, \cdots, a_m : \mathcal{H}_m\}$, where $a_j$ de-

[1]Note that $h$ can be any hash function, e.g. `hashCode()` of the Java `String` class, or `murmurHash` function available online at http://sites.google.com/site/murmurhash/.

notes the $j^{th}$ abstraction and $\mathcal{H}_j$ denotes the subset of hash features that are clustered together into $a_j$ based on some similarity measure. The nodes on the $m$-cut $\gamma_m$ correspond to *abstract features* in the low $m$-dimensional space.

The algorithm for learning AHs used in this work was introduced by Silvescu et al. (2009) and is reviewed below.

**Learning AHs over hash features.** The *input* of the algorithm is a set of hash features $\mathcal{H}$, along with their frequencies in each class $y_l \in \mathcal{Y}$, where $\mathcal{Y}$ represents the set of all classes. That is,

$$\left[h_i : \left[x_i^{(h,\mathcal{D})}, y_l\right]_{y_l \in \mathcal{Y}}\right]_{i=0,\cdots,b-1}. \quad (1)$$

Specifically, $\left[x_i^{(h,\mathcal{D})}, y_l\right]$ represents the number of times the $i^{th}$ hash feature co-occurs with the class $y_l$ in the training set $\mathcal{D}$. The *output* is an AH $\mathcal{T}$ over $\mathcal{H}$. The algorithm initializes each abstraction with a hash feature in $\mathcal{H}$, then recursively merges pairs of abstractions that are most "similar" to each other, and returns $\mathcal{T}$ after $b-1$ steps. The AH is stored in a last-in-first-out (LIFO) stack.

Two abstractions are considered "similar" if they occur within similar class contexts. The *class context of a hash feature* in $\mathcal{H}$ is defined as the conditional probability distribution of the class variable $Y$ (which takes values in $\mathcal{Y}$) given $h_i$, $p(Y|h_i)$, and is estimated from $\mathcal{D}$ as follows:

$$\hat{p}(Y|h_i) = \left[\frac{\left[x_i^{(h,\mathcal{D})}, y_l\right]}{\sum_{y_l \in \mathcal{Y}} \left[x_i^{(h,\mathcal{D})}, y_l\right]}\right]_{y_l \in \mathcal{Y}}. \quad (2)$$

Furthermore, the *class context of an abstraction* $a_j : \mathcal{H}_j$ is $p(Y|a_j)$, and is obtained using a weighted aggregation of the contexts of the hash features in $\mathcal{H}_j$. That is,

$$\hat{p}(Y|a_j) = \sum_{t=1}^{|\mathcal{H}_j|} \frac{x_t^{(h,\mathcal{D})}}{\sum_{r=1}^{|\mathcal{H}_j|} x_r^{(h,\mathcal{D})}} \cdot \hat{p}(Y|h_t), \quad (3)$$

where $h_t \in a_j$ for all $t = 1, \cdots, |\mathcal{H}_j|$. Again, $x_t^{(h,\mathcal{D})}$ represents the number of times $h_t$ occurs in $\mathcal{D}$.

The distance between two abstractions $a_i$ and $a_j$, denoted by $d(a_i, a_j)$, is defined as $d(a_i, a_j) = (p(a_i) + p(a_j))JS_{\pi_i,\pi_j}(p(Y|a_i), p(Y|a_j))$, where $JS_{\pi_i,\pi_j}(p(Y|a_i), p(Y|a_j)$ is the weighted Jensen-Shannon divergence[2] (Lin 1991) between $a_i$'s and $a_j$'s class contexts, with $\pi_i = \frac{p(a_i)}{p(a_i)+p(a_j)}$ and $\pi_j = \frac{p(a_j)}{p(a_i)+p(a_j)}$. The abstractions $a_i$ and $a_j$ with the smallest distance between their class contexts are merged into $a_k$ at each step.

The choice of the distance above based on the Jensen-Shannon divergence *explicitly* minimizes the reduction in mutual information between the features and the class variable. Let $A$ be a random variable that takes values in the set of abstractions on the cut $\gamma_m$. Silvescu et al. (2009) have shown that the reduction in mutual information between $A$ and $Y$ due to the merger $\{a_i, a_j\} \rightarrow a_k$ is given by $d(a_i, a_j)$. Hence, the choice of the distance $d$ to cluster features induces an ordering over the cuts $\gamma_m$ in $\mathcal{T}$ with the smallest reduction in mutual information from one cut to another. A cut $\gamma_m$ is *uniquely* obtained by removing $m-1$ elements from the top of the last-in-first-out stack. An array of indices of size $b$ (corresponding to the number of hash features) is used to specify the membership of hash features into the abstractions on the cut $\gamma_m$. The space complexity for storing this array is $O(b)$ that is presumably significantly smaller than $O(d)$, which is the space complexity for storing the array when only the abstraction is used.

Note that random clustering (or hashing) can suffer from loss of information due to hashing in the same cluster of two high frequency features with significantly different class distributions, whereas class-based clustering (or abstraction) with the Jensen-Shannon divergence, can avoid such a pitfall by not allowing features in the same cluster, unless the class distributions of the two features are significantly similar.

Using abstraction, mid-size dimensional hash vectors $\mathbf{x^h}$ of size $b$ are *abstracted* into lower dimensional vectors $\mathbf{x^a}$ of size $m$, with $m \ll b$. Specifically, each hash feature $h_i$ is mapped into the $j_{th}$ abstraction on the cut $\gamma_m$, to which it belongs. The index $j$ represents the index of the $i^{th}$ hash feature in $\mathbf{x^a}$, $j = 0, \cdots, m-1$. Frequency counts of the hash feature are stored at the corresponding index. Evaluating abstractions requires only an array entry look-up.

An entry $j$ in $\mathbf{x^a}$ records the frequency counts of hash features that are grouped together in an abstraction. That is, $x_j^a = \sum_{i:a(i)=j} x_i^h$, for $i = 0, \cdots, b-1$, $j = 0, \cdots, m-1$.

## Experiments and Results

We evaluated the combination of hashing and abstraction for classification on two data sets: the Reuters RCV1 data set of newswire stories (Lewis et al. 2004), and the Cora benchmark data set of research articles (McCallum et al. 2000).

For the Reuters RCV1 data set, we used the standard split of $781,265$ articles for training and $23,149$ articles for testing, and considered the binary classification of the most populous class in the data set, CCAT, which was also used by Shi et al. (2009) for feature hashing. For Cora, we used a subset[3], which consists of 3191 machine learning research articles found on the Web, and categorized into seven classes.

### Experimental Design

Our experiments are designed to explore the following questions: (i) What is the influence of the hash size on the performance of classifiers that use hash features, and what is the hash size at which the performance starts degrading? (ii) How effective is feature hashing on prohibitively high dimensional $n$-gram representations? (iii) How does the performance of feature abstraction compare to that of feature hashing and feature selection after an initial reduction is done by feature hashing to a mid-size dimensional space?

[2]The weighted Jensen-Shannon divergence between two probability distributions $p_i$ and $p_j$ with weights $\pi_i$ and $\pi_j$, is given by: $JS_{\pi_i,\pi_j}(p_i, p_j) = \pi_i KL(p_i||\bar{p}) + \pi_j KL(p_j||\bar{p})$, where $\bar{p}$ is the weighted average distribution, $\bar{p} = \pi_i p_i + \pi_j p_j$, and $KL(p_i||\bar{p})$ represents the Kullback-Leibler divergence between $p_i$ and $\bar{p}$.

[3]Available at http://www.cs.umd.edu/projects/linqs/projects/lbc/.

To answer these questions, we proceed with the following steps. We first preprocess the data by removing punctuation, and performing stemming. We did not remove stop words, as they are required to form $n$-grams from text documents. An $n$-gram was obtained by concatenating $n$ consecutive words.

Given a document $D$, we apply feature hashing in two settings as follows: (i) we first tokenize the document. Each word token is then *hashed* into a hash key. We refer to this setting as the "bag of words" approach (BoW); (ii) we tokenize the document and, in addition to the word tokens in $D$, we also generate all the bigrams. This setting uses both unigrams and bigrams. Each string, i.e., unigram or bigram, is *hashed* into a hash key. We refer to this setting as $n$-grams.

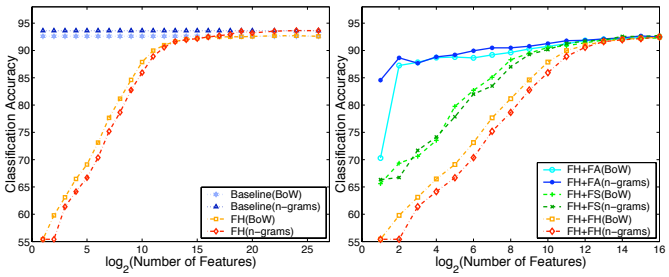
We train Support Vector Machine (SVM) classifiers (Fan et al. 2008) on hash features and investigate the influence of hash size on SVMs' performance. That is, we train SVMs on BoW and $n$-grams, denoted as FH(BoW) and FH($n$-grams), respectively, for values of the hash size ranging from a 1 bit hash (i.e., $2^1$) to a 26 bit hash (i.e., $2^{26}$), in steps of 1 (i.e., for all powers of 2 up to 26), and compare their performance.

Furthermore, we apply hashing to the sparse high dimensional BoW and $n$-gram representations to reduce the dimensionality to a mid-size $b$-dimensional space, e.g., $b = 2^{16}$ or $b = 2^{14}$, where the performance of SVMs starts to degrade due to hash collisions (this was guided by the experiment described above). We perform further dimensionality reduction using abstraction, selection, and hashing, and compare the performance of SVMs trained using feature abstraction, feature selection, and feature hashing, respectively. The features used in each case are the following:

- a bag of $m$ abstractions, over the available $b$ hash features, obtained using the combination of hashing and abstraction. This experiment is denoted by FH+FA.
- a bag of $m$ hash features chosen from the available $b$ hash features, using feature selection by mutual information. This experiment is denoted by FH+FS.
- a bag of $m$ hash features obtained using feature hashing over the available $b$ hash features, i.e., for each token, the first feature hashing produces an index $i$ such that $i = h(\text{token}) \ \% \ b$, whereas the second hashing produces an index $j$ such that $j = i \ \% \ m$. Note that for powers of 2, using hashing to reduce dimensionality from $d$ to $b$ and then from $b$ to $m$ is the same as reducing the dimensionality from $d$ to $m$, due to *modulo* operator properties. This experiment is denoted by FH+FH.

In experiments, for each of the above representations, we used the standard $tf$-$idf$ weighting scheme. For SVM, we used the LibLinear implementation[4]. As for the hash function, we experimented with both the `hashCode` of the Java `String` class, and the `murmurHash` function. We found that the results were not significantly different from one another in terms of the number of hash collisions and classification accuracy. We also experimented with both $\xi : \mathcal{S} \rightarrow \{\pm 1\}$ and $\xi \equiv 1$ (i.e., the actual counts), and found that the results were not significantly different. Thus, in the results shown next, we used the `hashCode` function and $\xi \equiv 1$.

[4]Available at http://www.csie.ntu.edu.tw/ cjlin/liblinear/




(a) (b)

Figure 1: Results on the Reuters RCV1 binary data set for both FH(BoW) and FH($n$-grams): (a) the influence of the hash size on the performance of SVMs; (b) the comparison of FH+FA with FH+FH and FH+FS.

| $b$ | Reuters RCV1 (BoW) | | Reuters RCV1 ($n$-grams) | |
|---|---|---|---|---|
| | # features | Collisions % | # features | Collisions % |
| $2^{26}$ | 360543 | 0.24 | 10570900 | 8.29 |
| $2^{24}$ | 357543 | 1.07 | 8327354 | 30.35 |
| $2^{22}$ | 346520 | 4.17 | 3925756 | 81.15 |
| $2^{20}$ | 305969 | 16.18 | 1048556 | 99.98 |
| $2^{18}$ | 195988 | 53.74 | 262144 | 100 |
| $2^{16}$ | 65237 | 97.63 | 65536 | 100 |
| $2^{14}$ | 16384 | 100 | 16384 | 100 |

Table 1: The number of unique features (denoted as # features) as well as the rate of collisions on the Reuters RCV1 data set for both BoW and $n$-grams representations.

On Reuters RCV1, we report the classification accuracy on the test set, whereas on Cora, we report the average classification accuracy obtained in a 5-fold cross validation experiment, along with the 95% confidence intervals. The accuracy is shown as a function of the number of features. The $x$ axis of all figures shows the number of features on a $\log_2$ scale (i.e., number of bits in the hash-table).

## Results on Reuters RCV1

Figure 1a shows the influence of the hash size $b$ on the performance of SVMs, trained using both FH(BoW) and FH($n$-grams) on Reuters RCV1, where $b$ range from $2^1$ to $2^{26}$.

**The influence of hash sizes on classifiers' performance.** As can be seen in the figure, for both FH(BoW) and FH($n$-grams), as the hash size $b$ increases, the performance of SVMs increases as well, due to a smaller rate of hash collisions. Table 1 shows, on Reuters RCV1, for both $n$-grams and BoW, the number of unique features and the percentage of collisions for various hash sizes. The number of unique features is calculated as the number of non-zero entries in the hash vector, and the number of collisions as the number of entries with at least one collision. Note that the percentage of collisions below $2^{14}$ is 100%.

As the hash size increases beyond $2^{16}$, the performance of SVMs does not change substantially, and, eventually, converges. Moreover, as the hash size increases beyond $2^{16}$, the percentage of hash collisions decreases until almost no col-

lisions occur (Table 1). The performance of SVMs trained on hash features in the $2^{26}$ dimensional space is matched by that of SVMs trained on hash features in the $2^{20}$ dimensional space, suggesting that hash collisions beyond $2^{20}$ does not significantly distort the data. Similar to Weinberger et al. (2009), we considered as *baseline*, models trained on hash features in the $2^{26}$ space. As $2^{26}$ highly exceeds the number of unique features, and the rate of hash collisions becomes close to zero, this can be seen as a fine approximation of the SVMs trained without hashing. Furthermore, we considered $2^{16}$ as the point where the performance starts degrading.

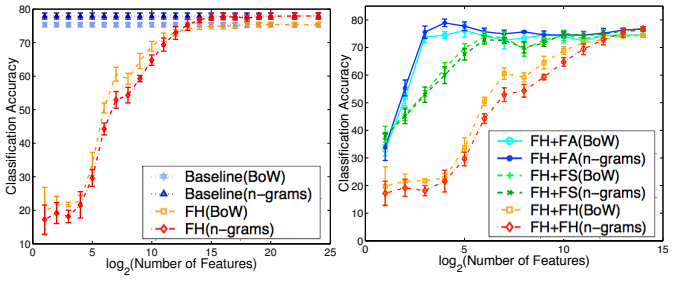
We conclude that, if hashing is used to reduce dimensionality from very high dimensions, e.g., $2^{26}$, to mid-size dimensions, e.g., $2^{16}$, hash collisions do not substantially hurt the classification accuracy, whereas if it is used to reduce dimensionality from mid-size to smaller dimensions, e.g., $2^{10}$, hash collisions significantly distort the data, and the corresponding SVMs result in poor performance (see Figure 1a).

**Comparison of FH(BoW) with FH($n$-grams).** Figure 1a also contrasts the performance of SVMs trained using FH(BoW) with that of their counterparts trained using FH($n$-grams). When the hash size is large, SVMs trained on FH($n$-grams) outperform those trained on FH(BoW), whereas for small hash sizes, SVMs trained on FH($n$-grams) perform worse than those trained on FH(BoW). As the total number of unique $n$-grams (words and bigrams), far exceeds the total number of unique words (BoW), for the same hash size, the percentage of hash collisions is higher in the case of $n$-grams (see Table 1). We conclude that feature hashing is very effective on prohibitively high-dimensional $n$-gram representations, which would otherwise be impractical to use, thus, resulting in memory-efficiency.

**Comparison of FH+FA with FH+FH and FH+FS.** Figure 1b shows the results of the comparison of FH+FA with FH+FH and FH+FS on Reuters RCV1, for both FH(BoW) and FH($n$-grams). As the performance of SVMs using hash features starts degrading substantially for hash sizes below $2^{16}$ (see Figure 1a), suggesting that the hash collisions start to significantly distort the data, we first reduce the dimensionality by feature hashing into the $2^{16}$ dimensional space. We further reduced the dimensionality by abstraction, selection, and hashing (as described in the previous subsection).

As can be seen in the figure, FH+FA makes it possible to train SVMs that use substantially smaller number of dimensions compared to the *baseline*, for a small drop in accuracy. For example, with $2^{10} = 1024$ hash size, the accuracy of SVM is $91.26\%$ using FH+FA($n$-grams) as compared to $93.62\%$ accuracy achieved by the *baseline*, with $2^{26} = 67,108,864$ hash size. As the hash size decreases, the performance of SVMs trained using FH+FA decreases much slower compared to that of SVMs trained using both FH+FS and FH+FH. For any choice of the hash size, SVMs trained using FH+FA outperform those trained using FH+FS and FH+FH. The performance of SVMs trained using FH+FA(BoW) and FH+FS(BoW) is not substantially different from that of SVMs trained using FH+FA($n$-grams) and FH+FS($n$-grams), respectively (Figure 1b).

We conclude that abstraction results in better performing models compared to hashing and feature selection after hashing was initially used to reduce the dimensionality of high-dimensional spaces to mid-size dimensional spaces.




(a) (b)

Figure 2: Results on the Cora multi-class data set for both FH(BoW) and FH($n$-grams): (a) the influence of the hash size on the performance of SVMs; (b) the comparison of FH+FA with FH+FH and FH+FS.

## Results on Cora

Figure 2a shows the results of the comparison of FH(BoW) with FH($n$-grams), and the impact of the hash size on the classification performance on Cora. Also on Cora, Figure 2b shows the results of the comparison of FH+FA with FH+FS and FH+FH. Furthermore, Table 2 shows the number of unique features as well as the rate of collisions on Cora for both BoW and $n$-grams representations. As expected, given this significantly smaller data set, the number of unique features is much smaller than that of Reuters RCV1, and consequently, the rate of collisions is much smaller. For this reason, we used as *baseline*, an SVM trained on hash features in the $2^{24}$ dimensional space. We also considered $2^{14}$ as the point where the performance starts degrading, and first reduced the dimensionality by feature hashing into the $2^{14}$ dimensional space, instead of $2^{16}$, as for Reuters RCV1. We performed FA, FS, and FH on the $2^{14}$ dimensional space.

As can be seen in the figures, the conclusions drawn on Reuters RCV1 hold for Cora as well. In addition, SVMs trained using FH+FA($n$-grams) significantly outperform the *baseline*, i.e., an SVM trained on FH($n$-grams) using a $2^{24}$ hash size. Hence, FH+FA can help minimize *overfitting* (through parameter smoothing) when the *labeled* training set is limited in size. FH+FA significantly outperforms FH+FS and FH+FH. However, FH+FS approaches FH+FA much faster compared to the results on Reuters RCV1, i.e., $2^6$.

## Conclusion

We presented an approach to reducing the dimensionality of sparse high-dimensional feature vectors. Our approach benefits from the advantages of both hashing and abstraction, by combining them in a coherent and principled manner. Specifically, hashing is used to reduce very high dimensions up to a point where the performance starts degrading, due to collisions of high-frequency features with highly divergent class distributions. Abstraction is further used to reduce these (hashed) dimensions to lower dimensions, by grouping

| $b$ | Cora (BoW) | | Cora ($n$-grams) | |
|---|---|---|---|---|
| | # features | Collisions % | # features | Collisions % |
| $2^{24}$ | 10662 | 0.03 | 152692 | 0.43 |
| $2^{22}$ | 10652 | 0.12 | 150653 | 1.77 |
| $2^{20}$ | 10606 | 0.55 | 142687 | 7.12 |
| $2^{18}$ | 10460 | 1.94 | 115836 | 26.67 |
| $2^{16}$ | 9827 | 8.04 | 59093 | 75.21 |
| $2^{14}$ | 7794 | 29.53 | 16383 | 99.92 |

Table 2: The number of unique features (denoted as # features) as well as the rate of collisions on the Cora data set for both BoW and $n$-grams representations.

together hash features with "similar" class distributions. The results of our experiments on two text corpora demonstrate the feasibility of our approach, which advances algorithms that can *efficiently* process sparse high-dimensional data into low-dimensional feature vectors at runtime. In the future, we plan to apply and integrate the approach presented here for fast classification and retrieval of short documents for social networks such as Twitter and Facebook.

## Acknowledgments

We would like to thank Doina Caragea and our anonymous reviewers for their constructive comments, which helped improve the presentation of this paper. This material is based in part on work supported by grants from The National Science Foundation and The Lockheed Martin Corporation. Any opinions, findings, and conclusions expressed here are those of the authors and do not necessarily reflect the views of the National Science Foundation and Lockheed Martin.

## References

Achlioptas, D. 2003. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *J. Comput. Syst. Sci.* 66:671–687.

Baker, D., and McCallum, A. 1998. Distributional clustering of words for text classification. In *Proc. of SIGIR-98*.

Berry, M. W. 1992. Large-scale sparse singular value computations. *Intl. Journal of Supercomputer Applications* 6:13–49.

Bingham, E., and Mannila, H. 2001. Random projection in dimensionality reduction: applications to image and text data. In *Proc. of the 7th ACM SIGKDD'01*, 245–250.

Blei, D.; Ng, A.; and Jordan, M. 2003. Latent Dirichlet allocation. *Journal of Machine Learning Research* 3:993–1022.

Deerwester, S.; Dumais, S. T.; Furnas, G. W.; Landauer, T. K.; and Harshman, R. 1990. Indexing by latent semantic analysis. *J. of the American Society for Inf. Science* 41(6):391–407.

desJardins, M.; Getoor, L.; and Koller, D. 2000. Using feature hierarchies in bayesian network learning. In *In Proc. of SARA '02*, 260–270.

Duda, R. O.; Hart, P. E.; and Stork, D. G. 2001. *Pattern Classification (2nd Edition)*. Wiley-Interscience.

Fan, R.-E.; Chang, K.-W.; Hsieh, C.-J.; Wang, X.-R.; and Lin, C.-J. 2008. LIBLINEAR: A library for large linear classification. *J. of Machine Learning Res.* 9:1871–1874.

Forman, G., and Kirshenbaum, E. 2008. Extremely fast text feature extraction for classification and indexing. In *Proc. of the 17th ACM CIKM*, 1221–1230.

Ganchev, K., and Dredze, M. 2008. Small statistical models by random feature mixing. In *Proceedings of the ACL-2008 Workshop on Mobile Language Processing*.

Gionis, A.; Indyk, P.; and Motwani, R. 1999. Similarity search in high dimensions via hashing. In *Proc. of the 25th VLDB Conference*, 518–529.

Golub, G., and van Loan, C. 1983. *Matrix Computations.* North Oxford Academic, Oxford, UK.

Guyon, I., and Elisseeff, A. 2003. An introduction to variable and feature selection. *J. Mach. Learn. Res.* 3:1157–1182.

Hofmann, T. 1999. Probabilistic latent semantic analysis. In *Proc. of UAI'99*.

Indyk, P., and Motwani, R. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proc. of the ACM Symp. on Theory of Computing*, 604–613.

Johnson, W., and Lindenstrauss, J. 1984. Extensions of lipshitz mapping into hilbert space. *Contemporary Mathematics, Amer. Math. Soc.* 26:189206.

Jolliffe, I. T. 1986. *Principal Component Analysis.* Springer-Verlag, New York.

Kang, D.-K.; Zhang, J.; Silvescu, A.; and Honavar, V. 2005. Multinomial event model based abstraction for sequence and text classification. In *In: Proceedings of SARA'05*.

Langford, J.; Li, L.; and Strehl, A. 2007. Vowpal wabbit online learning project.

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

Liberty, E.; Ailon, N.; and Singer, A. 2008. Dense fast random projections and lean walsh transforms. In *Proc. of the Intl. Workshop, APPROX '08 / RANDOM '08*, 512–522.

Lin, J. 1991. Divergence measures based on the shannon entropy. *IEEE Trans. on Information theory* 37:145–151.

McCallum, A.; Nigam, K.; Rennie, J.; and Seymore, K. 2000. Automating the contruction of internet portals with machine learning. *Information Retrieval J.* 3:127–163.

Papadimitriou, C. H.; Tamaki, H.; Raghavan, P.; and Vempala, S. 1998. Latent semantic indexing: a probabilistic analysis. In *Proc. of PODS*, 159–168.

Pereira, F.; Tishby, N.; and Lee, L. 1993. Distributional clustering of english words. In *Proc. of ACL*, 183–190.

Rahimi, A., and Recht, B. 2008. Random features for large-scale kernel machines. In *Proc. of NIPS*.

Roweis, S. 1998. Em algorithms for pca and spca. In *Proceedings of NIPS*, 626–632.

Shi, Q.; Petterson, J.; Dror, G.; Langford, J.; Smola, A.; Strehl, A.; and Vishwanathan, V. 2009. Hash kernels. In *Proc. of the Artificial Intelligence and Statistics, Florida 2009.*

Silvescu, A.; Caragea, C.; and Honavar, V. 2009. Combining super-structuring and abstraction on sequence classification. In *ICDM*, 986–991.

Slonim, N., and Tishby, N. 1999. Agglomerative information bottleneck. In *Proc. of NIPS*.

Tishby, N.; Pereira, F. C.; and Bialek, W. 1999. The information bottleneck method. In *Invited paper to The 37th annual Conf. on Communication, Control, and Comp.*, 368–377.

Weinberger, K.; Dasgupta, A.; Attenberg, J.; Langford, J.; and Smola, A. 2009. Feature hashing for large scale multitask learning. In *Proc. of ICML*.

Yang, Y., and Pederson, J. O. 1997. Feature selection in statistical learning of text categorization. In *In Proceedings of ICML*, 412–420.