Scalable Text Classification with Sparse Generative Modeling

Antti Puurula

Department of Computer Science, The University of Waikato, Private Bag 3105, Hamilton 3240, New Zealand

Abstract. Machine learning technology faces challenges in handling "Big Data": vast volumes of online data such as web pages, news stories and articles. A dominant solution has been parallelization, but this does not make the tasks less challenging. An alternative solution is using sparse computation methods to fundamentally change the complexity of the processing tasks themselves. This can be done by using both the sparsity found in natural data and sparsified models. In this paper we show that sparse representations can be used to reduce the time complexity of generative classifiers to build fundamentally more scalable classifiers. We reduce the time complexity of Multinomial Naive Bayes classification with sparsity and show how to extend these findings into three multi-label extensions: Binary Relevance, Label Powerset and Multi-label Mixture Models. To provide competitive performance we provide the methods with smoothing and pruning modifications and optimize model meta-parameters using direct search optimization. We report on classification experiments on 5 publicly available datasets for large-scale multi-label classification. All three methods scale easily to the largest available tasks, with training times measured in seconds and classification times in milliseconds, even with millions of training documents, features and classes. The presented sparse modeling techniques should be applicable to many other classifiers, providing the same types of fundamental complexity reductions when applied to large scale tasks.

Keywords: sparse modeling, multi-label mixture model, generative classifiers, Multinomial Naive Bayes, sparse representation, scalable computing, big data.

1 Introduction

Machine learning systems are operating on increasingly larger amounts of data, or "Big Data". A dominant idea has been to tackle these challenges by parallelizing algorithms with cluster computing and more recently cloud computing. An alternative approach to the scalability problem is to change the algorithms to more scalable ones. Most types of web data are naturally sparse, including graph and text data. The models can be made sparse as well. Sparse computing methods offer the possibility of solving the scalability problem by reducing the

P. Anthony, M. Ishizuka, and D. Lukose (Eds.): PRICAI 2012, LNAI 7458, pp. 458–469, 2012.

computational complexity of the algorithms themselves, offering fundamentally more efficient solutions.

Sparse computing works by representing data and models using sparse matrix representations. For example, a vector of word counts of a document $\boldsymbol{w}$ can be represented by two smaller vectors of indexes and non-zero counts. Alternatively a hash table can be used for this, for constant time lookups and additions. In both cases the complexity of storing sparse information is reduced from full $|\boldsymbol{w}|$ to sparse complexity $s(\boldsymbol{w})$, where $s(\boldsymbol{w})$ is the number of non-zero counts. Fundamental reductions in computing complexities can be gained by choosing the correct sparse representation.

In this paper we show that by using the correct sparse representations the time complexity of generative classifiers can be reduced. We propose a sparse time complexity algorithm for MNB classification. We then demonstrate sparse generative classification with three multi-label extensions of Multinomial Naive Bayes (MNB), representing baseline approaches to multi-label classification. Binary Relevance (BR) method extends MNB by considering each label in multi-label classification as a separate problem, performing binary-class classifications. Label Powerset (PS) converts each labelset to a label, performing multi-class classification. Finally, Multi-label Mixture Modeling (MLMM) decomposes labelsets into mixture components, performing full multi-label classification. For each method a couple of meta-parameters are optimized using a direct search algorithm to provide realistic performance on the datasets. The direct search optimizations are done using a parallelized random search algorithm, to optimize the microaveraged F-score of development sets for each method.

Five freely available large-scale multi-label datasets are used for the experiments, using reported preprocessing for comparison of results. It is demonstrated that the use of sparse computing results in training times measured in seconds and classification times in milliseconds, even on the largest datasets with millions of documents, features and classes.

The paper continues as follows. Section 2 proposes sparse computation with the MNB model. Section 3 proposes three extensions of sparse MNB to multi-label classification. Section 4 presents experimental results on the five datasets and Section 5 completes the paper with a discussion.

2 Sparse Computation with Multinomial Naive Bayes

2.1 Multinomial Naive Bayes

Naive Bayes (NB) models [1, 2, 3] are generative graphical models, models of the joint probability distribution of features and classes. In text classification the joint distribution $p(\boldsymbol{w}, m)$ is that of word count vectors $\boldsymbol{w} = [w_1, ..., w_N]$ and label variables $m : 1 \leq m \leq M$, where N is the number of possible words and M the number of possible labels. Bayes classifiers use the Bayes theorem to factorize the joint distribution into *label prior* $p(m)$ and *label conditional* $p_m(\boldsymbol{w})$ models with separate parameters, so that $p(\boldsymbol{w}, m) = p(m)p_m(\boldsymbol{w})$. NB uses the additional assumption that the label conditional probabilities are independent,

so that $p_m(\boldsymbol{w}) = \prod_n p_m(w_n, n)$. Multinomial Naive Bayes (MNB) parameterizes the label conditional probabilities with a Multinomial distribution, so that $p_m(w_n, n) \propto p_m(n)^{w_n}$. In summary, MNB takes the form:

$$p(\boldsymbol{w}, m) = p_m(\boldsymbol{w})p(m) \propto p(m) \prod_{n=1}^{N} p_m(n)^{w_n}, \tag{1}$$

where $p(m)$ is Categorical and $p_m(n)^{w_n}$ Multinomial.

2.2 Feature Normalization

Modern implementations of MNB use feature normalizations such as TF-IDF [4]. Surprisingly, this method developed for improving information retrieval performance has been shown to correct many of the incorrect data assumptions that the MNB makes [3]. The version of TF-IDF we use here takes the form:

$$w_n = \frac{\log[1 + w_n^u]}{s(\boldsymbol{w}^u)} \log[\max(1, \frac{D}{D_n} - 1)], \tag{2}$$

where w_n^u is the original word count, $s(\boldsymbol{w}^u)$ the number of non-zero counts in the word vector, D_n the number of training documents word n occurs in, and D the number of training documents.

The first factor (TF) in the function performs unique length normalization [5] and word frequency log transform. Unique length normalization is used, as it has been shown to be consistent across different types of text data [5]. The log transform corrects for the "burstiness effect" of word occurrences in documents, not captured by a Multinomial. As shown in [3], performing a simple log-transform corrects this relatively well. The second factor (IDF) performs an unsmoothed Croft-Harper IDF transform. This downweights words that occur in many documents and gives more importance to rare words. The Croft-Harper IDF downweights the common more words severely, actually setting the weight of words occurring in more than half the documents to 0. This induces sparsity and can be useful when scaling to large-scale tasks.

2.3 Sparse Representatation and Classification with Generative Models

It is a common practice in text classification to represent word count vectors $\boldsymbol{w}$ sparsely using two smaller vectors, one for indices of non-zero counts $\boldsymbol{v}$ and one for the counts $\boldsymbol{c}$. A mapping from dense to sparse vector representation can be defined $k(\boldsymbol{w}) = [\boldsymbol{v}, \boldsymbol{c}]$. Less commonly, the Multinomial models can be represented sparsely in the same way, using a vector for non-zero probability indices and one for the probabilities. By using the right type of sparse representations we can reduce both the space and time complexity of generative models much further.

Instead of having either a dense or sparse vector for each Multinomial, all Multinomial counts can be represented together in the same hashtable, using

tuples of indices $\{m, n\}$ as the key and the log-probability $\log(p_m(n))$ as the stored value. This is known as the *dictionary of keys* representation for sparse matrices. Like dense vectors, the counts can be updated and queried in constant time. Like sparse vectors, the space use of storing the Multinomial is $s(\boldsymbol{w})$. But unlike sparse vectors, there is no need for allocating vectors when a resize is needed, or when a new label is encountered in training data.

The dictionary of keys is an efficient representation for model training, but for classification an even more efficient sparse representation exists. The *inverted index* forms the core technique of modern information retrieval, but surprisingly it has been proposed for classification use only recently [6]. An inverted index can be used to access the multinomials, consisting of a vector $\boldsymbol{\kappa}$ of label lists called *postings lists* $\boldsymbol{\kappa}_n$. In this paper it is shown that the inverted index can be used to reduce the time complexity of inference with generative models.

A naive algorithm for MNB classification computes the probability of the document for each label and keeps the label maximizing this probability. Taking the data sparsity $s(\boldsymbol{w})$ into account, this has the time complexity $O(s(\boldsymbol{w})M)$. With the inverted index, we can substantially reduce this complexity. Given a word vector, only the labels occurring in the postings lists can be considered for evaluation. To avoid a classification error in abnormal cases, the probability for the apriori most likely label needs to be precomputed. The *evaluation list* $\boldsymbol{\varrho}$ of labels can be computed by taking a union of the occurring labels, $\boldsymbol{\varrho} = \cup_{n:w_n>0}\boldsymbol{\kappa}_n$. Replacing the full set of labels with the evaluation list results in sparse $O(s(\boldsymbol{w})|\boldsymbol{\varrho}|)$ time complexity.

When conventional smoothing methods such as Dirichlet prior or interpolation are used, the smoothing probabilities can be precomputed and only updated for each label. Using this with generative modeling, we get another time complexity reduction. When constructing the evaluation list, the matching words between the word vector and unsmoothed multinomial can be saved as *update lists* $\boldsymbol{\nu}_m$ for each label. This reduces the time complexity to $O(s(\boldsymbol{w}) + \sum_{m\in\boldsymbol{\varrho}} |\boldsymbol{\nu}_m|))$ where $|\boldsymbol{\nu}_m|$ are the update list sizes, at worst $\min(s(\boldsymbol{w}), s(p_m^u))$. Algorithm 1 gives a pseudocode description of sparse MNB classification.

3 Multi-label Extensions of Naive Bayes

Multi-label classification deals with the extension of single-label classification from a single label to set of labels, or equivalently a binary labelvector of label occurrences $\boldsymbol{l} = [l_1, ..., l_M]$. In supervised multi-label tasks the training dataset is labeled with the labelsets. Evaluation is done using a variety of metrics, most commonly the micro-averaged and macro-averaged F-scores [7]. We optimize and evaluate using micro-averaged F-score as the measure, as this been most commonly used with the text datasets we are experimenting on.

We evaluate three scalable extensions to MNB for sparse multi-label classification. The first two are the problem transformation methods of Binary Relevance(BR) [8] and Label Powerset(PS) [9], that are commonly used as baselines in multi-label classification [10, 11]. The third one is a multi-label mixture model

Algorithm 1. Sparse MNB Classification

$log\_smooth = 0$
for all $n \in k(\boldsymbol{w})_1$ **do** ▷ Iterate document words $k(\boldsymbol{w})_1$
  $log\_smooth+ = \log(p^s(n)) * w_n$
  for all $m \in \boldsymbol{\kappa}_n$ **do** ▷ Iterate postings list $\boldsymbol{\kappa}_n$
    $\boldsymbol{\nu}_m = \cup(\boldsymbol{\nu}_m, (n))$
$m^{max} = m^{apriori}$
$p^{max} = \log(p(m^{apriori})) + log\_smooth$
for all $m \in \boldsymbol{\varrho}$ **do** ▷ Iterate evaluation list $\boldsymbol{\varrho}$
  $p^{new} = \log(p(m)) + log\_smooth$
  for all $n \in \boldsymbol{\nu}_m$ **do** ▷ Iterate update list $\boldsymbol{\nu}_m$
    $p^{new}+ = (\log(p_m(n)) - \log(p^s(n))) * w_n$
  if $p^{new} > p^{max}$ **then**
    $p^{max} = p^{new}$
    $m^{max} = m$

(MLMM) that we have developed, that uses mixture modeling to decompose label combinations.

3.1 Binary Relevance

The binary relevance method [8] considers each label in the labelvector independently, by performing a binary classification for each label. The advantages of BR are that it is very efficient and easy to implement with any classifier capable of binary-label classification. The disadvantage is that it totally ignores label correlations, in the worst cases classifying all labels as positive or none. A relevance thresholding scheme is commonly used to improve the results, by adjusting the relevance decision boundary to maximize the evaluation score on a held-out development set. Extending the MNB classifier for BR is straightforward, with a positive Multinomial and a corresponding negative Multinomial for each label. Since we work with very large numbers of labels, we can approximate the negative Multinomial with a background distribution $p(n)$ with little loss in accuracy.

3.2 Label Powerset

The label powerset method [9] is a straightforward way to perform multi-label classification with single-label classifiers. Each labelset in training is converted to a single label identifier and converted back to labelsets after classification. Both operations can be done using hash table lookups, externally to the classifier. The main disadvantage is the increased space and time complexity of classification. Instead of models for at most M labels, PS constructs models for each labelset configuration $\boldsymbol{l}$ occurring in the training data. It neither takes into account similarities between the labelsets, and can only classify labelsets that are seen in

the training data. Despite the theoretical problems, PS forms the basis of some of the most successful multi-label classification algorithms.

3.3 Multi-label Mixture Model

Multi-label mixture models [12, 13, 14] attempt generalization of MNB to multi-label data by decomposing labelset-conditional Multinomials into mixtures of label-conditional Multinomials. Here we propose a simple multi-label mixture model that is closely related to the original Multi-label Mixture Model [12] and Parametric Mixture Model [13], taking the form:

$$p(\boldsymbol{w}, \boldsymbol{l}) \propto p(\boldsymbol{l}) \prod_{n=1}^{N} [\sum_{m=1}^{M} \frac{l_m}{s(\boldsymbol{l})} p_m(n)]^{w_n} \tag{3}$$

By constraining the labelvector to a single label $s(\boldsymbol{l}) = 1$, the model reduces to MNB. A number of choices exist for modeling $p(\boldsymbol{l})$, one being a Categorical distribution over the labelvectors [12]. We use a fixed mixture of a Categorical with a smoothing distribution:

$$p(\boldsymbol{l}) = 0.5p^u(\boldsymbol{l}) + 0.5p^s(s(\boldsymbol{l})), \tag{4}$$

where p^u is the unsmoothed Categorical, p^s is a Categorical over labelcounts $s(\boldsymbol{l})$ corrected to $\boldsymbol{l}$ event space.

Classification with multi-label mixture models requires heuristics in practice, as a naive algorithm would perform a 2^M enumeration of all the possible labelsets. A common greedy algorithm [12, 13] starts with a labelvector of zeros and iteratively sets to 1 the label m that improves $p(\boldsymbol{w}, \boldsymbol{l})$ the most. The iteration ends if no label improves $p(\boldsymbol{w}, \boldsymbol{l})$. The labelcount prior p^s constrains what would be M^2M to a maximum of Mq^2 evaluations, where $q = \max_{\boldsymbol{l}:p(\boldsymbol{l})>0}(s(\boldsymbol{l}))$ is the largest labelcount in training data, resulting in $O(q^2\, M\, s(\boldsymbol{w}))$ complexity.

This algorithm can be improved by taking into account the sparse improvements we've proposed for MNB and some additional heuristics. We can add caching of probabilities, so that in each iteration the probabilities $p(w_n)$ can be updated instead of recomputed. This reduces the time complexity to $O(Mqs(\boldsymbol{w}))$. We can also remove all non-improving labels in each iteration from the evaluation list as a weak heuristic. Finally, we can combine these with the sparse classification done in Algorithm 1, to get the worst time complexity of $O(q(s(\boldsymbol{w}) + \sum_{m \in \boldsymbol{\varrho}} |\boldsymbol{\nu}_m|))$, or simply q times the sparse MNB complexity.

3.4 Model Modifications

For competitive performance and to deal with realistic data some modifications are required in generative modeling. An example of a mandatory modification is smoothing for the conditional Multinomials. In this paper we use four meta-parameters $\boldsymbol{a}$ to produce realistic performance for each compared method, except for the additional label thresholding meta-parameter a_5 used with BR.

The first modification is the *conditional smoothing*. We use Jelinek-Mercer interpolation, or linear interpolation with a background model. For each method, we estimate a background Multinomial concurrently to the Multinomials and interpolate this with the conditionals, so that $p_m(n) = (1 - a_1)p_m^u(n) + a_1 p^s(n)$.

The second modification is to enable training with limited memory to very large datasets. We first constrain the hashtable for the conditionals to a maximum of 8 million counts, so that adding keys above that is not allowed. In addition we use pruning with the IDF weights. When a count is incremented, we compare its IDF-weighted value to an *insertion pruning* threshold a_2. If the value is under the threshold, the count is removed from the hashtable. When stream training is used, the IDF-values can be approximated with running estimates, giving gradually more accurate pruning.

A third modification is to scale the priors, replacing $p(\boldsymbol{l})$ by $p(\boldsymbol{l})^{a_3}$. This is commonly done in speech recognition, where language models are scaled. We've added this modification as it has a very considerable effect, especially in cases where the prior is less usable for a dataset and the generative model still weights labels according to the prior.

As a fourth modification we add a pruning criteria to the classification algorithm. We add a sorting to the evaluation lists, by scoring each label as the sum of matching TF-IDF weighted word counts $q_m = \sum_n w_n \mathbf{1}_{p_m^u(n)>0}$, and sorting the evaluation list by the scores q_m. We can then use the ranked evaluation list, by stopping the classification once the mean log probability of the evaluated labels deviates too far from the maximum log probability found, $mean\_logprob - a_4 < max\_logprob$. The scoring adds a $O(|\boldsymbol{\varrho}| \log(|\boldsymbol{\varrho}|))$ term to the time complexity, but this does not increase the worst time complexity in typical cases.

3.5 Meta-parameter Optimization

We use a direct search [15] approach for optimizing the meta-parameters. The function value $f(\boldsymbol{a})$ we optimize is the micro-averaged F-score of held-out development data. The type of direct search we use is a random search algorithm [16, 17], an approach best suited for the low-dimensional, noisy and multimodal function we are dealing with.

In random search the current best point $\boldsymbol{a}$ is improved by generating a new point $\boldsymbol{d} \leftarrow \boldsymbol{a} + \boldsymbol{\Delta}$ with step $\boldsymbol{\Delta}$ and replacing $\boldsymbol{a}$ with $\boldsymbol{d}$ if the new point is as good or better, if $f(\boldsymbol{a}) \leq f(\boldsymbol{d})$, $\boldsymbol{a} \leftarrow \boldsymbol{d}$. The iterations are then continued to a maximum number of iterations I. A number of heuristics are commonly used to improve basic random search. We use a couple common ones, along a novel Bernoulli-Lognormal function for step generation.

Instead of generating a single point, we generate a population of J points in order to fully use parallel computing. Instead of having a single point for generation, we keep many, so that if any point $\boldsymbol{d}_j \leftarrow \boldsymbol{a} + \boldsymbol{\Delta}_j$ improves or equals $f(\boldsymbol{a})$, we replace the current set of points by the Z best points sharing the best value. Subsequent points are then generated evenly from the current set of best points $\boldsymbol{a}_{1+j\%Z}$.

We generate steps with a Bernoulli-Lognormal function, so that for each a_t we first generate a step direction with a uniform Bernoulli $b_{jt} \in (-1, 1)$ and then multiply the stepsize by a Lognormal $e^{\mathcal{N}(0,2)}$. We combine this with a global adaptive stepsize decrease, so that we start stepsizes at half range $c_t = 0.5 * (max_t - min_t)$ and multiply them by 1.2 after an improving iteration and by 0.8 otherwise. This gives the step generation process as $\Delta_{jt} \leftarrow b_{jt}\, c_t\, e^{\mathcal{N}(0,2)}$. In addition we improve variance among the generated points by generating each alternate step in a mirrored direction: if $j\%2 = 0$, $\boldsymbol{b}_j \leftarrow -\boldsymbol{b}_{(j-1)}$. A heuristic starting point $\boldsymbol{a}$ is used and to reduce the search space each meta-parameter is constrained to a suitable range $d_{jt} \leftarrow min(max(a_{1+j\%Z} + \Delta_{jt}, min_t), max_t)$ found in model development.

4 Experiments

4.1 Experiment Setup

The experiment software was implemented using Java and executed on single thread on a 3.40GHz Intel i7-2600 processor using 4GB of RAM. Five recent large-scale multi-label datasets were used for the experiments, all freely available for download. Table 1 shows the dataset statistics. The numbers of distinct labelsets are omitted from the table, but in the worst case this is 1468718 classes for the WikipL dataset, close to a million and a half. The datasets were preprocessed by lowercasing, stopwording and stemming, and stored in LIBSVM sparse format.

Table 1. Statistics for the five training datasets. Documents D in training, unique labels M, unique words N, mean of unique labels per document $e(s(\boldsymbol{l}))$ and mean of unique words per document $e(s(\boldsymbol{w}))$.

Dataset	Train D	Labels M	Words N	$e(s(\boldsymbol{l}))$	$e(s(\boldsymbol{w}))$	Task description
RCV1-v2	343117	350	161218	1.595	63.169	News articles
Eurlex	17381	3870	179540	5.319	**270.346**	Legal documents
Ohsu-trec	197555	14379	291299	**12.389**	81.225	Medical abstracts
DMOZ	392756	27874	593769	1.028	174.124	Web pages
WikipL	**2363436**	**325014**	**1617125**	3.262	42.534	Wikipedia articles

The datasets consist of WikipL, DMOZ, Ohsu-trec, Eurlex and RCV1-v2, each split to a training set, held-out development set and evaluation set. WikipL and DMOZ are the larger datasets from LSHTC2[1] evaluation of multi-label classification. For Eurlex[2] [18] the first 16381 documents of eurlex_tokenstring_CV1-10_train.arff were used as the training set, the last 1000 as the development set and eurlex_tokenstring_CV1-10_test.arff as the evaluation set. Ohsu-trec[3] used

[1] http://lshtc.iit.demokritos.gr/

[2] http://www.ke.tu-darmstadt.de/resources/eurlex

[3] http://trec.nist.gov/data/t9_filtering.html

both the title and abstract as document text, preprocessed by lowercasing, removing <3 letter words and using the Porter stemmer. The MeSH terms were used as labels with additional specifiers to terms discarded. Ohsumed.88-91 is used as the training dataset, the first 1000 documents of ohsumed.87 as the development set and the rest as the evaluation set. The files lyrl2004_tokens_test_pt*.dat for RCV1-v2[4] [19] were used as the training set, the first 1000 documents of lyrl2004_tokens_train.dat were used as the development set and the last 8644 as the evaluation set. The categorization rcv1-v2.industries.qrels was used for labeling.

4.2 Experiment Results

The three evaluated methods were optimized for micro-averaged F-score on each development set using a 50x30 (50 iterations, 30 points) random search. Figure 1 compiles the results from the runs, showing training times in seconds, development set classification times in milliseconds and evaluation set micro-averaged F-scores. The time estimates were computed as the median of 8 runs. Table 2 shows the same numbers in a table form in addition to the development set F-scores.

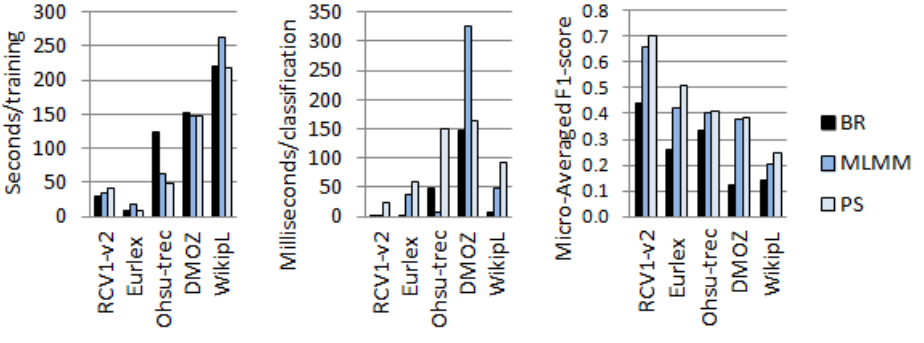

Fig. 1. Median training times, median classification times and micro-averaged F-scores

One-tailed paired t-tests were used to test the differences in F-scores and times, verified by Wilcoxon signed rank tests with $p < 0.05$. Significances from t-tests $p < 0.05$ are shown in parenthesis. BR is outperformed in accuracy by both MLMM ($p < 0.006$) and PS ($p < 0.004$). In addition the effect size is very large, with BR falling behind by almost half the F-score on average. The difference between MLMM and PS accuracy is not significant, although on average PS is over 3% F-score better than MLMM. In terms of training set times all models perform similarly, with no significant differences and very similar mean times. In terms of

[4] http://www.daviddlewis.com/resources/testcollections/rcv1/

classification times the large variance causes only the difference between BR and PS to be significant ($p < 0.013$), although the mean times suggest BR is twice as fast as both MLMM and PS, and MLMM is somewhat faster than PS.

Table 2. Median training times, median classification times and micro-averaged F-scores

(a) Training times in seconds

	BR	MLMM	PS
RCV1-v2	30.73	35.53	42.50
Eurlex	9.56	18.60	9.01
Ohsu-trec	123.51	63.43	48.32
DMOZ	152.95	148.29	147.14
WikipL	219.82	261.92	218.01

(b) Classification times in ms

	BR	MLMM	PS
RCV1-v2	0.25	2.18	24.73
Eurlex	1.71	38.47	59.70
Ohsu-trec	47.72	5.90	150.85
DMOZ	148.55	324.79	164.98
WikipL	5.96	49.18	91.21

(c) Development set micro-averaged F-scores

	BR	MLMM	PS
RCV1-v2	0.439	0.660	**0.702**
Eurlex	0.259	0.422	**0.508**
Ohsu-trec	0.332	0.405	**0.407**
DMOZ	0.121	0.381	**0.383**
WikipL	0.143	0.206	**0.249**

(d) Evaluation set micro-averaged F-scores

	BR	MLMM	PS
RCV1-v2	0.434	0.665	**0.705**
Eurlex	0.242	0.408	**0.498**
Ohsu-trec	0.318	**0.402**	0.401
DMOZ	0.101	**0.362**	0.358
WikipL	0.111	0.190	**0.228**

5 Discussion

This paper showed how sparse matrix representations can be applied to reduce the complexity requirements of generative models. Although sparse representations such as the inverted index are fundamental in fields such as information retrieval, we have found no prior work on explicitly applying sparse representations for reducing space and time complexities for probabilistic models. It is likely that sparse representations are used in practical implementations of existing models, but the connections to complexity theory have been so far omitted.

We demonstrated how generative classifiers such as MNB can utilize sparse representations for reducing the time complexity of classification. We then used these representations with three extensions of MNB on the largest publicly available multi-label classification datasets. To get representable performance, a couple of parameterized modifications were used. The meta-parameters required by these were optimized regarding the micro-averaged F-score of development sets with a direct search algorithm. All 3 classifiers could be trained in some minutes on a single processor, with millions of documents, features and classes. Although not optimized with classification speed in mind, the 3 classifiers performed classifications in times ranging from microseconds to some hundreds of milliseconds, even when using over a million classes.

In the experiments presented here no comparisons to dense classifiers were made, as it would be tedious to compare dense classification speeds with the large

datasets. In preliminary work we attempted several toolkits, but did not find ones that could scale to the databases discussed here. For future research it will be interesting to see what other classifiers can benefit from sparse representations. This can potentially change what classifiers are preferred in large scale tasks. It is expectable that the use of sparse representations becomes a mainstay of machine learning with the processing of web-scale tasks.

References

[1] Maron, M.E.: Automatic indexing: An experimental inquiry. J. ACM 8, 404–417 (1961)
[2] McCallum, A., Nigam, K.: A comparison of event models for Naive Bayes text classification. In: AAAI 1998 Workshop on Learning for Text Categorization, pp. 41–48. AAAI Press (1998)
[3] Rennie, J.D., Shih, L., Teevan, J., Karger, D.R.: Tackling the poor assumptions of naive bayes text classifiers. In: ICML 2003, pp. 616–623 (2003)
[4] Jones, K.S.: A Statistical Interpretation of Term Specificity and its Application in Retrieval. Journal of Documentation 28(1), 11–21 (1972)
[5] Singhal, A., Buckley, C., Mitra, M.: Pivoted document length normalization. In: Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 1996, pp. 21–29. ACM, New York (1996)
[6] Shanks, V.R., Williams, H.E., Cannane, A.: Indexing for fast categorisation. In: Proceedings of the 26th Australasian Computer Science Conference, ACSC 2003, vol. 16, pp. 119–127. Australian Computer Society, Inc., Darlinghurst (2003)
[7] Tsoumakas, G., Katakis, I., Vlahavas, I.P.: Mining multi-label data. In: Maimon, O., Rokach, L. (eds.) Data Mining and Knowledge Discovery Handbook, pp. 667–685. Springer (2010)
[8] Godbole, S., Sarawagi, S.: Discriminative methods for multi-labeled classification, pp. 22–30 (2004)
[9] Boutell, M.R., Luo, J., Shen, X., Brown, C.M.: Learning multi-label scene classification. Pattern Recognition 37(9), 1757 (2004)
[10] Tsoumakas, G., Katakis, I., Vlahavas, I.: A Review of Multi-Label Classification Methods. In: Proceedings of the 2nd ADBIS Workshop on Data Mining and Knowledge Discovery, ADMKD 2006, pp. 99–109 (2006)
[11] Read, J., Pfahringer, B., Holmes, G., Frank, E.: Classifier Chains for Multi-label Classification. In: Buntine, W., Grobelnik, M., Mladenić, D., Shawe-Taylor, J. (eds.) ECML PKDD 2009, Part II. LNCS, vol. 5782, pp. 254–269. Springer, Heidelberg (2009)
[12] McCallum, A.: Multi-label text classification with a mixture model trained by EM. In: Proceedings of the AAAI 1999 Workshop on Text Learning (1999)
[13] Ueda, N., Saito, K.: Parametric mixture models for multi-labeled text. In: Advances in Neural Information Processing Systems, vol. 15, pp. 721–728. MIT Press (2002)
[14] Wang, H., Huang, M., Zhu, X.: A generative probabilistic model for multi-label classification. In: Proceedings of the 2008 Eighth IEEE International Conference on Data Mining, pp. 628–637. IEEE Computer Society, Washington, DC (2008)
[15] Powell, M.J.D.: Direct search algorithms for optimization calculations. Acta Numerica 7, 287–336 (1998)

[16] Favreau, R.R., Franks, R.G.: Statistical optimization. In: Proceedings Second International Analog Computer Conference (1958)
[17] Brunato, M., Battiti, R.: Rash: A self-adaptive random search method. In: Cotta, C., Sevaux, M., Sörensen, K. (eds.) Adaptive and Multilevel Metaheuristics. SCI, vol. 136, pp. 95–117. Springer (2008)
[18] Loza Mencía, E., Fürnkranz, J.: Efficient Multilabel Classification Algorithms for Large-Scale Problems in the Legal Domain. In: Francesconi, E., Montemagni, S., Peters, W., Tiscornia, D. (eds.) Semantic Processing of Legal Texts. LNCS, vol. 6036, pp. 192–215. Springer, Heidelberg (2010)
[19] Lewis, D.D., Yang, Y., Rose, T.G., Li, F.: RCV1: A New Benchmark Collection for Text Categorization Research. J. Mach. Learn. Res. 5, 361–397 (2004)