Available online at www.sciencedirect.com
ScienceDirect
Neurocomputing 70 (2007) 1466–1481
NEUROCOMPUTING
www.elsevier.com/locate/neucom

One-class document classification via Neural Networks ☆

Larry Manevitz[a,b,*], Malik Yousef[a,c]

[a]Department of Computer Science, University of Haifa, Haifa, Israel
[b]Department of Experimental Psychology, Institute of Mathematics, Oxford University, Oxford, UK
[c]Wistar Institute, University of Pennsylvania, Philadelphia, Pennsylvania, USA

Received 28 September 2004; received in revised form 12 May 2006; accepted 18 May 2006
Communicated by T. Heskes
Available online 10 October 2006

Abstract

Automated document retrieval and classification is of central importance in many contexts; our main motivating goal is the efficient classification and retrieval of "interests" on the internet when only positive information is available. In this paper, we show how a simple feed-forward neural network can be trained to filter documents under these conditions, and that this method seems to be superior to modified methods (modified to use only positive examples), such as Rocchio, Nearest Neighbor, Naive-Bayes, Distance-based Probability and One-Class SVM algorithms.

A novel experimental finding is that retrieval is enhanced substantially in this context by carrying out a certain kind of uniform transformation ("Hadamard") of the information prior to the training of the network.

Keywords: Classification; Automated document retrieval; Feed-forward neural networks; Machine learning; One-class classification; Autoencoder; Bottleneck neural network

1. Introduction

The goal of this research is to develop a filter that can examine a corpus of documents and choose those of interest.

This requires a method of defining what it means to be "of interest" and a method of matching the documents to this definition. It is natural and convenient to assume that the definition of interest be learned (see also [24,30,28]) by observing examples, and, in this context, it is pertinent to assume only positive examples. That is, one can have a sample set of examples of documents which are "interesting" and from this set develop a filter which can be applied to select other such "interesting" documents. The reason for using only positive examples is that one can (i) obtain such examples simply by observation; i.e. for many applications an "active" teacher will not be necessary (ii) in many contexts, it is easier to find "typical" examples rather than typical "non-examples". By "typical" we mean a valid sampling which appropriately represents the space. See [20,2,12,11,27] for other papers on the use of positive examples only. For example, in trying to develop an intelligent "web-browser", one can imagine a system that inobstrusively tracks a user and by following documents of interest to him, builds a filter for future automated retrieval. (See [33,18] for the use of the ideas of this paper in building such an automated filter.)

In this paper, we investigate the efficacy of such methods isolated to the context of document retrieval; but because of the motivations above, we restrict ourselves as much as possible to positive information.

In the sections below, we report on experiments with various methods some using "pure" positive information; and some "contaminated" by using negative examples albeit *after* the training stage. Our methods involve focusing on applying the "bottleneck" or "autoencoder" neural network in different ways. Our methods are also

☆This work was partially supported by *HIACS*, the Haifa University Interdisciplinary Center for Advanced Computer Science, and the Neurocomputation Laboratory located at the Caesarea Rothschild Institute for Interdisciplinary Computer Science.
*Corresponding author.
E-mail addresses: manevitz@cs.haifa.ac.il (L. Manevitz), yousef@cs.haifa.ac.il (M. Yousef).

0925-2312/$ - see front matter © 2006 Published by Elsevier B.V.
doi:10.1016/j.neucom.2006.05.013

compared with other information retrieval methods adapted to the positive example case.

2. The neural networks classifier

The basic design of the filter under discussion in this section is a feed-forward neural network. In order to incorporate the restriction of positive examples only, we used the design of a feed-forward network with a "bottleneck". That is, under the assumption that the documents are represented in an m-dimensional space; we choose a three level network with m inputs, m outputs and k neurons on the hidden level, where $k<m$. The network is then trained, under standard backpropagation to learn the identity function on the sample examples. (This design, now usually called the "autoencoder" after Japkowicz et al. [12] was first used by Munro et al. [7] to produce a compression algorithm. See also [12] for another use as a novelty detector.)

The overall idea is that while the bottleneck prevents learning the full identity function on m-space; the identity on the small set of examples is in fact learnable. Thus, the set of vectors for which the network acts as the identity function is a kind of sub-space which is similar to the trained set. (This avoids the "saturation" problem of learning from only positive examples.) Thus, the filter is defined by applying the network to a given vector; if the result is the identity, then the vector is "interesting".[1] See Fig. 1.

To apply this idea to documents, one needs to (i) decide on the number of hidden neurons in the architecture (i.e. the tightness of the bottleneck) and choose the appropriate learning rates, etc. related to any application of backpropagation [29], (ii) encode the documents as vectors, (iii) determine the appropriate acceptance thresholds (i.e. how close to the identity is required for acceptance) when applying the trained networks to classify new documents.

2.1. Architecture and training parameters

The architecture is feed forward with three levels. There were 20 real valued inputs and outputs and six hidden level neurons in most of our experiments. All neurons were standard sigmoids. Initial weights were chosen as small random values. The positive example vectors (see below) were divided into two sets, training and testing. Training proceeded according to standard backpropagation with learning parameter .75 and momentum coefficient .08 until the mean-square error fell below a predetermined level. These parameters are standard (i.e. the same as in [11]; the results were not sensitive to small changes in these parameters).

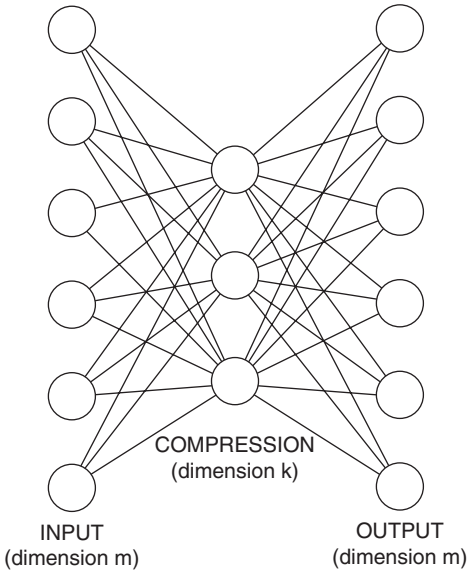

Fig. 1. A neural network with bottleneck.

Experimentally six hidden level neurons gave the best result. We also examined our neural network filter with different sizes of input and output (corresponding to different numbers of features from the document, see below) in order to investigate how this influenced the performance of the classification task. (See Table 9 and the appendix.) In addition, we examined different choices and sizes of the hidden level; both in choosing the compression of the bottleneck and considering the possibility of expansion [10] instead of compression. (See the appendix for these details.)

2.2. Data set and measurements

Our experiments were performed on the standard *Reuters* data-base [16], a preclassified collection of short articles. This is one of the standard test beds used to test information retrieval algorithms [6,9].

For each category, we first used 25% of the positive data for training and then tested on the remaining data (positive and negative). Since other authors [6,13,19] have used the "ModApte" split which is the opposite split (75–25%), we reran the experiments with this split as well and report the results in the appendix. The results are quite similar to those reported in this section.

Table 1 shows the 10 most frequent categories along with the number of training and test examples in each. We treated each of the 10 categories as a binary classification task and evaluated the classifiers for each category separately.

We used two kinds of measures to assess the performance: (1) the number accepted divided by the total number of documents in the class, and (2) the F_1 measure, the recall and the precision values.

[1] Although, intuitively, it is more natural to use a bottleneck, recent work by Japkowicz [10] has used an autoencoder with an "expander"; i.e. with more neurons in the hidden level. We tried this as well, but with substantially poorer results. The Diablo [27] model uses a slightly different architecture than ours for autoencoder purposes.

Table 1
Number of training/test items

Category name	Num train	Num test
Earn	966	2902
Acquisitions	590	1773
Money-fx	187	563
Grain	151	456
Crude	155	465
Trade	133	401
Interest	123	370
Ship	65	195
Wheat	62	186
Corn	62	184

For text categorization, the effectiveness measure of recall and precision are defined as follows:

$$\text{recall} = \frac{\text{Number of items of category identified}}{\text{Number of category members in test set}},$$

$$\text{precision} = \frac{\text{Number of items of category identified}}{\text{Total items assigned to category}}.$$

Van Rijsbergen [31] defined the F_1-measure as a combination of recall (R) and precision (P) with an equal weight in the following form: $F_1(\mathrm{R},\mathrm{P}) = 2RP/(R+P)$.

Note that this measure is in fact sensitive to the amount of "balance" between the positive and negative examples used in evaluation. Accordingly, we also calculate (and present in the appendix) the accuracy; i.e. the percentage of positive and negative examples that are correctly classified. (See [25] for more on "balance".)

2.3. Text representation and feature selection

The simplest representation one can imagine, is to assume a dictionary of possible words in all documents, and then to include a binary vector in $\{0,1\}^n$ whose dimension n is the number of words in the dictionary. Thus, a vector with a 1 in dimension j means that word w_j is in the document.

It is clear that one would not want to use simple variants of words, but there are well known algorithms to strip simple suffixes, etc. [23]. If instead of a binary vector, one counts the number of occurrences in the document, and then normalizes the vector to one, one has the term *frequency* representation. In the literature, a correction to this vector is often felt to be necessary to account for the number of documents in which the word occurs. (That is, the importance of a word in a specific text is inversely related to how often it appears in other documents.)

Thus, instead of the frequency, one uses the *tf-idf* (term-frequency-inverse-document-frequency) representation [26] which is given by the following formula (where $f(word)$ means the frequency of the word in the document and $N(word)$ means the number of documents the word appears in):

$$tfidf(word) = f(word) \cdot \left[\log \frac{n}{N(word)} + 1\right].$$

To explain the heuristic mix of neural network encoding and heuristic choice of representation used in our work, we will need a few definitions:

Let C, the "*corpus*" be the set of documents to be classified. Let T be a subset of C the class of "interesting" documents. Let E be a subset of T, the positive examples. The problem is to define a function (or "filter"), using only information from E that distinguishes T from $\overline{T}$, the complement of T.

We proceed as follows: let D be the *dictionary* of all words in $\bigcup E$; with each word associated to its frequency in the list. Heuristically, we eliminate words which occur in less than three documents (document frequency is less than 3); and use standard algorithms to (i) eliminate stop words and (ii) strip grammatical endings from common words [23].

From this dictionary, we then choose the m words that appear in the greatest number of documents of E. We call these "keywords"; however they are chosen automatically. The choice of m is rather arbitrary. In our examples we have typically used $m = 20$; this was influenced by the work of Weiner et al. [32].

However, we point out that the analysis in [32] used the "relevance" factor to rank the words. (See [32] for a definition.) This information is not available to us, if we assume knowledge of only positive examples. Table 9 shows some comparisons with different choices of m. There is not a clear indication of optimal size of m from this data. The proper choice of m in general requires further analysis.

(We point out here, that, while common [34] , this choice of features can be problematic. In particular, a substantial portion of examples can have very few non-zero features, making it very difficult for excellent recall without faulty precision. The distribution of number of non-zero features amongst the examples used are given in Table 2).

We performed additional experiments using very large vectors. (See the appendix.) In general, additional features can in fact improve the network although the improvement is very marginal.

For later reference (see "Hadamard product" below) we define v_T as the m-dimensional vector consisting of the frequencies of occurrence of the keywords over the training set of the positive examples. Then for each document $\mathbf{e} \in E$ we associate a vector of dimension m, which we will continue to designate as $\mathbf{e}$. Here e_i is the frequency of the ith chosen keyword in the document $\mathbf{e}$.

At this point, the natural way to proceed would be to use the vectors $\mathbf{e}$ to train the network by backpropagation to the identity function. Then the network could be used as a filter on the remainder of C, by (i) representing each document in C by the vector of frequencies (normalized to 1) of the chosen keywords (Note that the representation of each document in C depends on the class T for acceptance.)

Table 2
Distribution of examples by the number of features with non-zero value

#Non-zero	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Earn	10	13	18	79	115	175	128	112	99	85	66	38	18	7	3	0	0
Acq	17	28	53	50	87	76	57	54	52	43	27	18	10	6	7	3	2
Money	2	5	11	12	18	26	23	21	16	8	12	6	11	8	5	1	2
Grain	1	6	6	16	16	23	28	14	15	8	7	3	3	2	1	0	1
Crude	1	5	8	9	14	12	16	17	13	14	8	13	5	9	4	4	2
Trade	1	4	3	6	4	9	12	12	11	12	9	10	13	7	6	8	2
Int	1	2	8	3	21	10	17	16	18	17	4	2	2	0	0	2	0
Ship	2	5	13	11	4	5	7	5	5	6	1	0	1	0	0	0	0
Wheat	0	0	1	6	8	8	9	13	5	5	2	1	1	0	1	1	0
Corn	1	0	1	5	2	2	15	6	8	4	4	3	5	4	0	1	0

#Non-zero is the number of features with non-zero values for the example represented by the vector.

and (ii) applying the network to these vectors. If the result is sufficiently close to the identity the document is accepted. In the tables below, we refer to this as the "frequency representation".

2.4. Hadamard product

We discovered experimentally that the following additional transformation, H_T, of vectors substantially enhances performance.[2]

Here,

$$H_T(e_i) = e_i \cdot v_{Ti},$$

or

$$H_T(\mathbf{e}) = \begin{pmatrix} e_1 \\ e_2 \\ . \\ . \\ . \\ e_m \end{pmatrix} \otimes \begin{pmatrix} v_{T_1} \\ v_{T_2} \\ . \\ . \\ . \\ v_{T_m} \end{pmatrix} = \begin{pmatrix} e_1 \cdot v_{T_1} \\ e_2 \cdot v_{T_2} \\ . \\ . \\ e_m \cdot v_{T_m} \end{pmatrix}$$

i.e. take the component-wise product with the frequency vector of the dictionary.

Table 3 shows the comparison between using and not using this Hadamard product in two sample classes. Note that the Hadamard representation is substantially superior in avoiding false positives.

It seems reasonable to look for a Bayesian explanation of this phenomenon. Let g represent a given document, w_i the ith word in the dictionary, and E a set of training examples. Assume that e_i represents $P(w_i|g)$ and v_{T_i} represents $P(w_i|E)$; i.e. the probabilities of a chosen word being w_i given that you are either in the document being tested or in the class of interesting examples. One can then argue (under certain independent assumptions) using Bayes rule that the product vector is representing the adjusted probability of a word being chosen given that it is both in the document g and an interesting example. (See the appendix for a full derivation.)

In any case, it is clear that the Hadamard product emphasizes the differences between large and small feature entries.

[2]We call this the "Hadamard Product" after the well known term-wise matrix multiplication (sometimes also called the "Shur Product") and the term-wise product of infinite series.

Table 3
Neural networks comparison of the Hadamard and frequency representations for two typical categories in a neural network (using acceptance method 3)

Training/test set	Earn		Grain	
	Hadamard	Frequency	Hadamard	Frequency
	736/966 2322/2902	779/966 2339/2902	75/151 180/456	58/151 162/456
Earn	LT	LT	139/3868	49/3868
Acquisitions	329/2362	2316/2362	29/2359	66/2359
Money	98/750	733/750	3/749	15/749
Grain	49/607	599/607	LT	LT
Crude	65/620	613/620	15/619	41/619
Trade	57/534	495/534	27/515	39/515
Interest	67/493	447/493	9/493	13/493
Ship	8/260	260/260	3/234	2/234
Wheat	22/248	246/248	3/3	2/3
Corn	25/246	241/246	2/3	1/3
F_1	0.781292	0.41804	0.415704	0.379836
Recall	0.800138	0.805996	0.394737	0.355263
Precision	0.763314	0.282181	0.439024	0.408060

LT denotes that the topic is the "learned" topic; it is used in the training set and test set. After training is completed, the results of the training set and control test subset of the category are listed in the first two lines of the table. For each of the other categories, all examples which were cross-listed in the trained category were removed.

2.5. Acceptance threshold determination

We used (in different experiments) six methods of determining the threshold of acceptance when using the bottleneck network. Recall that the error of the neural network is the average of the square error (from the

identity function) over all the training examples. (Bottom Line: Method 6(b) is the final method of choice; obtaining essentially the best results and using only positive information in setting the threshold.)

1. The network was trained to a fixed constant error level (0.00625) determined in advance. The threshold is $error + 0.2sd$ where

   error = the average of the square error over all the training examples

   and

   sd = standard deviation of the average error.

   (This method uses only positive information.)
2. After training the network to an acceptable level of error (0.001 average square error); we then ranked the members of the training set according to their individual errors; and set the threshold at the error level of the 90th percentile. Note that this means that one should expect false negatives at the rate of 10%. (This method uses only positive examples.)
3. This is a more sophisticated method, based on a combination of variance and calculating the optimal F_1 measure. During training, we checked the F_1 values of the test set using different levels of error both as the stopping point and as the threshold. We discovered that there was *always* an error level at which the F_1 started a steep decline. (Note that the calculation of F_1 values requires the availability of negative examples albeit only after training of the network is completed. That is, the negative examples are used for setting the threshold only. In later methods below, this need of negative examples is eliminated.) See Fig. 2.
   We chose this error level to stop training. (It is our conjecture that this is the point where the network starts to reorganize itself to accept all data; i.e. starting to have many false positives.) Then a secondary analysis was performed to determine an optimal real multiple of the standard deviation of this average error to serve as the threshold.
4. In the fixed recall level approach, we predetermine a set of recall levels at which we want to compute precision, and analyze the ranked documents in the test set to determine, for each category, what decision threshold would lead to the desired set of recall levels at the precision–recall break-even points. (The precision–recall break-even point is defined as the value where the precision and recall value are equal.) Then the threshold is chosen as determined by the largest such computed break-even point.
   As in the previous method, this calculation requires knowledge of negative examples prior to using the test set, but not in the training set.
5. After running our experiments a heuristic way to determine the threshold using only positive information was suggested by Japkowicz et al. [12]. Here it is essentially suggested to train the network for a heuristically determined fixed number of epochs (e.g. 200); to discover the maximal error *in each epoch*; to find the threshold that is determined by these maximal errors; and then to relax it by, e.g. 25%. A new example is then accepted if it passes *all* epochs (after an initial period) with the relaxed threshold.
   However, direct application of this suggestion gave very poor results (i.e. the precision and hence the F_1 values dropped drastically). This was surprising given the excellent results reported in [12].
6. After consideration, we decided that the problem probably lies in the problematic representation of data in our study. That is, from Table 2, we see that there are, in some categories, a large number of examples which are almost the zero vector. Thus, we decided to implement Japkowicz' idea, but with the opposite tack; i.e. to *tighten* the threshold by an amount heuristically related to the percentage of near zero vectors in the training set. (The idea is that attempting to include these vectors must harm the precision; since they do not have enough information to distinguish them from negative examples.)
   (a) We then implemented this modified algorithm, using a tightening dependent on the statistics of Table 2. The result was a very dramatic increase in accuracy, yielding, in many ways, the best results of all methods.
   (b) We then tried a further modification, combining ideas from method 2 with the above, which removes the necessity to analyze each data set separately.
   In each epoch, separately, we tightened the threshold sufficiently to disallow the classification of the highest 25 percentile error cases from the training set. Again, the motivation is that there were many near zero vectors in our training set. The results were nearly as good as the previous method. We emphasize that this method does not use any negative information.

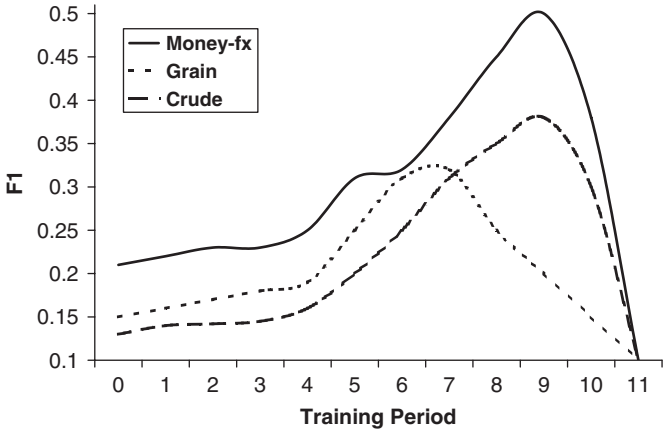

Fig. 2. F_1 values at different levels of error.

2.6. Comparison amongst variants of the bottleneck neural networks

In Table 3, we present the results on two typical classes using the third acceptance threshold determination presented above (using standard deviation and F_1 measure) for both the Hadamard and the frequency representations. In each entry the number of documents accepted and the total number of documents appears. Note that for the articles in the chosen category, it is desirable to accept as many items as possible; while for the articles in the other categories, it is desirable to accept as few items as possible.

It is clear from this table that the Hadamard representation is substantially superior to the frequency representation, especially in terms of avoiding false positives.

Table 4 shows a comparison of the *tf-idf* representation with the Hadamard representation according to the second method (90th percentile) of acceptance. Again we see that the Hadamard representation is superior, with the *tf-idf* giving many false positives.

In Table 5, we compared the third and fourth methods of acceptance for the 10 most frequent categories. The results show no significant difference between the two methods of acceptance.

In Table 6 we compare methods 5, 6(a) and (b). The results here were quite interesting. First, and initially surprisingly, method 5 was rather poor in this context. The relaxation of the threshold imposed in that method, while allowing almost perfect recall, had very poor precision, thereby resulting in poor F_1 performance. In essence, the relaxation allowed almost all data to be accepted into the class. We believe there are two major reasons for this failure: (1) the data is human classified real data; therefore, the classification is not 100% reliable; (2) the representation of the data is not efficacious. Table 2 shows that about 1.5% of the training data is in fact the zero vector in this representation, 4% has at most one non-zero dimension, 9% has at most two non-zero dimensions, 16% has at most three non-zero dimensions and 27.5% has at most four non-zero dimensions.

Table 4
Neural networks comparison of the *tf-idf* and Hadamard representations for three typical categories (using acceptance method 2)

Train test	Silver		Yen		Dollar	
	Had.	*tf-idf*	Had.	*tf-idf*	Had.	*tf-idf*
	12/14 13/16	12/14 16/16	25/28 21/32	25/28 2/32	69/77 66/93	69/77 11/93
Coffee	1/80	80/80	1/80	13/80	1/80	4/80
Earn	2/594	594/594	2/594	486/594	12/594	12/594
Jobs	0/73	73/73	0/73	12/73	3/73	17/73
Gold	31/123	123/123	3/123	54/123	5/123	17/123
Dollar	0/173	173/173	119/173	129/173	LT	LT
Yen	0/62	62/62	LT	LT	44/62	12/62
Silver	LT	LT	N/A	N/A	N/A	N/A
F_1	0.8125	0.02488	0.71186	0.00505	0.7333	0.1358

LT denotes that the topic is the "learned" topic; it is used in the training set and test set. (N/A denotes not available.)

Table 5
Neural networks comparison between standard deviation (method 3) and break-even point (method 4) for threshold determination

Category	Standard deviation			Break-even point		
	F_1	R	P	F_1	R	P
Earn	0.781	0.800	0.763	0.782	0.799	0.765
Acq	0.534	0.598	0.483	0.535	0.599	0.483
Money	0.542	0.641	0.470	0.540	0.600	0.491
Grain	0.415	0.394	0.439	0.413	0.399	0.428
Crude	0.537	0.505	0.573	0.539	0.498	0.587
Trade	0.573	0.600	0.547	N/A	N/A	N/A
Interest	0.496	0.416	0.616	N/A	N/A	N/A
Ship	0.393	0.328	0.492	0.377	0.297	0.517
Wheat	0.507	0.446	0.588	0.492	0.500	0.484
Corn	0.310	0.451	0.236	0.307	0.298	0.316

N/A denotes not available.

It is reasonable to assume that some of these vector representations are too weak to distinguish between data and should really be considered as noise. This means that setting a threshold that allows all of these data points to be accepted is too weak and this is what happened under method 5. (Japkowicz et al. [12] obtained excellent results under method 5. The number of non-zero dimensions of the training data used there is unknown to us, although we imagine it was not small.)

This analysis suggested pursuing the opposite tack from method 5; i.e. tightening the threshold requirement instead of relaxing it. Experimentation with the degree of tightening revealed that this does improve the results; however, the amount of optimal tightening varied substantially between the classification task, with larger tightening needed for the larger classes.

Method 6(a) presents a result obtained by these methods. In some ways this was the best result. The amount of tightening applied in this case was 15% for Earn, 5% for Acq, 3% for Money, Grain, Crude, Trade and Int; and .8% for Ship, Wheat and Corn. The different percentages correspond to the relative size of the training sets. (See Table 2.)

Since the analysis of 6(a) showed that changing the threshold to eliminate acceptance of some of the training set varies from classification to classification, we decided a logical and more uniform method would be to (1) rank the training set in each epoch according to error (2) tighten the threshold to reject the worst 25 percentile of data. This is method 6(b). The results here were essentially as successful as 6(a).

In Table 7, we compare method 1 (fixed threshold), method 3 (F_1 measure) and method 6(b). Method 3 and 6(b) were substantially superior to method 1, except for subcategory "earn" where method 6(b) was substantially

Table 6
Comparison between threshold determination methods 5, 6(a), and (b)

Category	Method 5			Method 6(a)			Method 6(b)		
	F_1	R	P	F_1	R	P	F_1	R	P
Earn	0.486	1.0	0.321	0.869	0.794	0.960	0.714	0.577	0.938
Acq	0.326	0.998	0.194	0.477	0.646	0.377	0.621	0.486	0.860
Money	0.111	1.0	0.059	0.529	0.486	0.579	0.642	0.621	0.665
Grain	0.094	0.982	0.049	0.424	0.342	0.557	0.473	0.368	0.661
Crude	0.099	0.997	0.052	0.602	0.572	0.636	0.534	0.544	0.524
Trade	0.079	1.0	0.041	0.436	0.396	0.486	0.569	0.546	0.595
Interest	0.073	1.0	0.038	0.638	0.613	0.665	0.487	0.337	0.874
Ship	0.039	1.0	0.019	0.353	0.471	0.283	0.361	0.512	0.278
Wheat	0.050	0.989	0.025	0.593	0.623	0.565	0.404	0.290	0.666
Corn	0.038	1.0	0.019	0.285	0.652	0.182	0.324	0.353	0.299
Avg	0.139	0.996	0.081	0.520	0.559	0.529	0.513	0.463	0.636
Wt.avg	0.296	0.997	0.186	0.637	0.648	0.662	0.615	0.516	0.795

Avg (also called *macro Avg*) is the simple average over all categories; *Wt. Avg* (also called *micro Avg*) takes into account the number of examples in the different test sets.

Table 7
Neural networks comparison between constant threshold using only positive examples (method 1), F_1 and standard deviation using negative examples (method 3) and the threshold tightening on percentile modification (method 6(b))

	Method 1			Method 3			Method 6b		
	F_1	R	P	F_1	R	P	F_1	R	P
Earn	0.408	0.793	0.274	0.781	0.800	0.763	0.714	0.577	0.938
Acq	0.533	0.650	0.451	0.534	0.598	0.483	0.621	0.486	0.860
Money	0.481	0.479	0.483	0.542	0.641	0.470	0.642	0.621	0.665
Grain	0.359	0.458	0.295	0.415	0.394	0.439	0.473	0.368	0.661
Crude	0.433	0.524	0.369	0.537	0.505	0.573	0.534	0.544	0.524
Trade	0.395	0.486	0.332	0.573	0.600	0.547	0.569	0.546	0.595
Int	0.278	0.729	0.171	0.496	0.416	0.616	0.487	0.337	0.874
Ship	0.226	0.384	0.160	0.393	0.328	0.492	0.361	0.512	0.278
Wheat	0.435	0.462	0.411	0.507	0.446	0.588	0.404	0.290	0.666
Corn	0.304	0.440	0.232	0.310	0.451	0.236	0.324	0.353	0.299
Avg	0.385	0.540	0.317	0.508	0.517	0.520	0.513	0.463	0.636
Wt.avg	0.428	0.651	0.336	0.614	0.638	0.600	0.615	0.516	0.795

The Hadamard representation was used.

superior on precision. There is no clear difference between 6(b) and 3. Note that method 6(b) uses only positive information whereas method 3 needs some negative examples to determine the threshold. On the other hand, the percentile reduction of method 6(b) was chosen heuristically.

In Table 8, we compare representations of the data. Two methods, Hadamard and frequency, are presented and their respective F_1, recall and precision values are presented for some of the ten most frequent categories in the database. These experiments were run using the third (F_1 and standard deviation) method of acceptance. The results show the clear superiority of the Hadamard representation.

In Table 9, using only the Hadamard representation, we investigated the effect of increasing the dimension of the features. (That is, allowing a larger number of keywords, while keeping the size of the hidden level the same.) There is some improvement but it is not dramatic.

3. Experimental comparisons

To test the validity of our approach and for comparison purposes, we implemented variants appropriate to the positive example case of the following algorithms: (1) Prototype (Rocchio) (2) Nearest Neighbor (3) Naive Bayes (4) Distance Based Probability (5) One-Class SVM. (Algorithms 2–4 were used by Datta [4] in his Ph.D. dissertation.)

3.1. Prototype (Rocchio's) algorithm

The Prototype algorithm is widely used in information retrieval ([21,1,22,15] and others). This algorithm is used

Table 8
Neural networks comparison of Hadamard and frequency representation

	Hadamard			Frequency		
	F_1	R	P	F_1	R	P
Earn	0.781	0.800	0.763	0.418	0.805	0.282
Acq	0.534	0.598	0.483	0.347	0.363	0.332
Money	0.542	0.641	0.470	0.475	0.420	0.546
Grain	0.415	0.394	0.439	0.379	0.355	0.408
Crude	0.537	0.505	0.573	0.476	0.410	0.566
Trade	0.573	0.600	0.547	0.536	0.513	0.561
Int	0.496	0.416	0.616	0.478	0.405	0.583
Ship	0.393	0.328	0.492	0.388	0.400	0.376
Wheat	0.507	0.446	0.588	0.414	0.430	0.400
Corn	0.310	0.451	0.236	0.315	0.434	0.247
Avg	0.508	0.517	0.520	0.422	0.453	0.430
Wt.avg	0.614	0.638	0.600	0.413	0.555	0.373

Table 9
F_1 values as a function of the number of features in the neural network (method 3) using the Hadamard representation

Category	Number of features				
	20	40	60	100	200
Grain	0.415	0.504	0.515	0.518	0.539
Crude	0.537	0.591	0.565	0.588	0.583
Trade	0.573	0.605	0.616	0.602	0.603
Interest	0.496	0.504	0.497	0.528	0.510
Ship	0.393	0.252	0.276	0.302	0.259
Wheat	0.507	0.487	0.491	0.501	0.495

frequently because it is considered a baseline algorithm and it is simple to implement. For extended details see [14].

The basic idea of the algorithm is to represent each document **e** as a vector in a vector space so that documents with similar content have similar vectors. The value e_i of the ith keyword is represented as the *tf-idf* weight.

The Prototype algorithm learns the class model by combining document vectors into a prototype vector c. This vector is generated by adding the document vectors of all documents in the class E

$$c = \sum_{e \in E} e.$$

In addition to generating the prototype vector, the Prototype algorithm learns a constant δ which is the maximum distance between the vectors in E and the prototype vector c. The distance is calculated using the cosine measure.

During classification, whenever the distance of a test example from the prototype vector is less than $\lambda\delta$ $(0 < \lambda <= 1)$ the test example is predicted as a member of the class.

In our experiments we estimated the optimal λ by testing λ over values between 0 and 1 and examining the F_1 value. For this classifier algorithm we store the prototype vector c, δ and λ.

3.2. Nearest neighbor

We used a modification of the Nearest Neighbor algorithm [4] to learn from positive examples for one class. The input for the modified algorithm, NN-PC (nearest neighbor positive class), are examples from only one class. In addition to storing the examples, NN-PC learns a constant δ which is the maximum distance that a test example can be to any learned example and still be considered a member of the positive class. Any test example that has a distance greater than $\lambda\delta$ from any training example will not be considered a member of the positive class. δ is calculated by

$$\delta = \max_{x} \min_{y \neq x} distance(x, y),$$

where x and y are two examples of the positive class, and the cosine measure is used as the distance function. Intuitively, δ records how much the examples vary from each other. When classifying test examples, if the test example varies "too much" from the positive examples then the test example is classified to not be a member of the positive class. More specifically, if $\exists x : distance(x, test) < \delta$ then the test example is classified as a member of the positive class, otherwise it is not.

In our experiments we estimated the optimal λ as we did in the Prototype algorithm. So for this classifier algorithm we store all the training vectors, δ and λ.

3.3. Naive Bayes

Traditional Naive Bayes calculates the probability of being in a class given an example, consisting of specific values of different attributes. One calculates this by assuming the different attributes are independent, applying Bayes' theorem and using the a priori probability of the different classes. According to [5] this is surprisingly accurate even when the independence assumption is broken. (See Domingos and Pazzani [5] for a discussion of the magnitude of the error. See also Hummel and Manevitz [8] for another "partial independence" assumption.)

However, when only the positive information is available, the usual calculations cannot be performed. Datta [4] showed how the algorithm can be modified for positive data only and we follow his presentation. Here we only state the parameters used in our implementation and refer the reader to [4] for explanations.

We calculate $p(d|E)$ as the product of $p(w|E)$ for all words in the dictionary that appear in the document d. Each of the $p(w|E)$ is estimated independently using the formula:

$$p(w|E) = \frac{n_w + 1}{n + |dictionary|},$$

where n_w is the number of times word w occurs in E, and n is the total number of words in E.

We calculate a threshold δ by the minimum over all examples in E, of the value $p(d|E)$ for each document in the set of examples. Then we experiment with values $\lambda\delta$ for $0<\lambda<=1$ as in the previous algorithms using F_1 to find the optimal threshold for acceptance. That is, given a new document d, we accept it if the calculated value $p(d|E)$ is larger than the determined $\lambda\delta$.

For this classifier algorithm we store δ and λ.

3.4. Distance Based Probabilities

This algorithm, DBP, establishes a distance between two documents, based on the appearance or the absence of a word from the dictionary in the document. Thus, the

$$distance(d_1, d_2) = \sum_{\text{w in difference set}} p(w|E)^2,$$

where the difference set is all words in the dictionary that appear in one document and not in the other; and $p(w|E)$ is calculated as in the previous section.

Another alternative, DBP-cosine, which proves to be superior in this context, replaces the distance with the cosine between two documents.

Then, as in the Nearest Neighbor algorithm, δ is computed in the same manner as above. The DBP also uses the same test for class membership as Nearest Neighbor, namely if $\exists x : distance(x, test)<\delta$ then the test example is classified to be a member of the positive class, otherwise it is not. Results are presented in Table 11.

In our experiments we estimated the optimal λ as we did in the Prototype algorithm. For this classifier algorithm we store all the training vectors, δ and λ.

3.5. One-class SVM

Scholkopf et al. [3] suggested a method of adapting the SVM methodology to the one-class classification problem. Essentially, after transforming the feature via a kernel, they treat the origin as the only member of the second class. Then using "relaxation parameters" they separate the image of the one class from the origin. Then the standard two-class SVM techniques are employed. Full details may be found in [17].

We also implemented a somewhat different version of the one-class SVM where we decided to identify these outliers by counting the features of an example with non-zero value; if this is less than a threshold then the feature is labeled as a negative example. One then continues with the standard two-class SVM. Full details may be found in [17].

3.6. Comparisons between the Neural Network and other algorithms

In this section we present the results using the different algorithms described above. In Table 10, we summarize the results from the Nearest Neighbor algorithm with Hadamard frequency *tf-idf* and binary document representation. The results show that the Hadamard representation is superior also in this context.

In Table 11, we summarize the results from the Naive Bayes algorithm and two version of the Distance Based Probability algorithm. The results show that using the cosine measure in the latter algorithm results in a dramatic improvement.

Our main comparisons between the different algorithms is summarized in Table 12. Seven methods are represented with their respective F_1 results. Note that except for method 6(b) and One-Class SVM, all use some negative examples after the training is completed to optimize their performance. NN under method 6(b) (column 2) uses only positive information. The bottleneck Neural Networks is presented with its Hadamard representation under methods 3 and 6(b) for the determination of threshold, Naive Bayes is presented with the probability representation,

Table 10
Nearest neighbor algorithm

	Hadamard			Frequency			*tf-idf*			Binary		
	F_1	R	P	F_1	R	P	F_1	R	P	F_1	R	P
Earn	0.703	0.856	0.596	0.719	0.782	0.665	0.694	0.866	0.580	0.707	0.831	0.615
Acq	0.476	0.727	0.354	0.480	0.714	0.362	0.486	0.714	0.368	0.498	0.719	0.381
Money	0.468	0.735	0.343	0.374	0.712	0.253	0.400	0.809	0.266	0.537	0.582	0.499
Grain	0.333	0.403	0.284	0.259	0.660	0.161	0.237	0.743	0.141	0.389	0.375	0.404
Crude	0.392	0.632	0.284	0.241	0.404	0.172	0.215	0.425	0.144	0.253	0.404	0.184
Trade	0.441	0.588	0.353	0.199	0.775	0.114	0.391	0.583	0.294	0.451	0.496	0.414
Int	0.295	0.337	0.263	0.307	0.651	0.201	0.180	0.727	0.102	0.329	0.443	0.262
Ship	0.389	0.338	0.458	0.239	0.358	0.179	0.190	0.353	0.130	0.301	0.374	0.252
Wheat	0.566	0.677	0.486	0.202	0.532	0.125	0.119	0.225	0.081	0.373	0.537	0.285
Corn	0.168	0.391	0.010	0.087	0.733	0.046	0.076	0.728	0.040	0.079	0.657	0.042
Avg	0.423	0.568	0.352	0.310	0.632	0.227	0.298	0.617	0.214	0.391	0.541	0.333
Wt.avg	0.531	0.706	0.431	0.490	0.705	0.407	0.481	0.736	0.376	0.531	0.671	0.451

Comparison between Hadamard, Frequency, *tf-idf* and Binary representation using only positive examples.

Table 11
Results for Naive Bayes and Distance Based Probability Algorithms

	Naive Bayes			DBP			DBP-cosine		
	F_1	R	P	F_1	R	P	F_1	R	P
Earn	0.708	0.709	0.706	0.471	0.912	0.318	0.612	0.846	0.479
Acq	0.503	0.609	0.428	0.312	0.926	0.187	0.376	0.798	0.246
Money	0.493	0.666	0.391	0.102	0.888	0.054	0.345	0.373	0.322
Grain	0.382	0.432	0.342	0.081	0.842	0.042	0.335	0.239	0.564
Crude	0.457	0.455	0.458	0.117	0.681	0.064	0.240	0.393	0.172
Trade	0.483	0.620	0.395	0.071	0.855	0.037	0.445	0.581	0.361
Int	0.394	0.481	0.334	0.072	0.859	0.037	0.326	0.675	0.215
Ship	0.288	0.271	0.308	0.033	0.846	0.017	0.231	0.184	0.310
Wheat	0.288	0.392	0.228	0.188	0.118	0.468	0.732	0.779	0.690
Corn	0.254	0.418	0.182	0.032	0.793	0.016	0.081	0.478	0.044
Avg	0.425	0.505	0.377	0.147	0.772	0.124	0.372	0.534	0.340
Wt.Avg	0.548	0.607	0.509	0.290	0.865	0.194	0.453	0.684	0.369

Table 12
Comparison of Neural Networks (Hadamard representation), Naive Bayes, Nearest Neighbor (Hadamard representation), Distance Based Probability (cosine measure), Prototype algorithms (*tf-idf* representation) and one-class SVM

	Neural Networks (method 3)	Neural Networks (method 6(b))	Naive Bayes	Nearest Neighbor	DBP-cosine	Prototype	One-class SVM radial basis
	F_1	F_1	F_1	F_1	F_1	F_1	F_1
Earn	0.781	0.714	0.708	0.703	0.612	0.637	0.676
Acq	0.534	0.621	0.503	0.476	0.376	0.468	0.482
Money	0.542	0.642	0.493	0.468	0.345	0.484	0.514
Grain	0.415	0.473	0.382	0.333	0.335	0.402	0.585
Crude	0.537	0.534	0.457	0.392	0.240	0.398	0.544
Trade	0.573	0.569	0.483	0.441	0.445	0.557	0.597
Int	0.496	0.487	0.394	0.295	0.326	0.454	0.485
Ship	0.393	0.361	0.288	0.389	0.231	0.370	0.539
Wheat	0.507	0.404	0.288	0.566	0.732	0.262	0.474
Corn	0.310	0.324	0.254	0.168	0.081	0.230	0.298
Avg	0.508	0.513	0.425	0.423	0.372	0.426	0.519
Wt.avg	0.613	0.615	0.547	0.530	0.453	0.516	0.572

Nearest Neighbor is presented with its Hadamard representation, Distance Based Probability is presented with the cosine measure, Prototype is presented with the *tf-idf* representation, the bottleneck Neural Network is also presented under method 6(b), and the One-Class SVM is presented with its binary representation using the Radial Basis as the kernel function. The results show that the Neural Network (under either representation) is superior to the other methods. However, the SVM methods turned out to be quite sensitive to the choice of representation and kernel in ways which are not well understood. See [17] for full details on the usage of the SVM methods for this problem.

4. Combining classifiers algorithms

If one assumes that the different algorithms function better on different parts of the data, it is reasonable to try a "high-level" combination of the algorithms. In this section, we performed a few such experiments combining three algorithms in different ways: the NN under method 3, the Naive Bayes Algorithm and the Nearest Neighbor algorithm.

We determined the threshold parameter δ for each algorithm separately. However, in estimating the optimal λ value, we used the F_1 values of the combined algorithm.

Table 13 summarizes the results for combining the Naive Bayes and the Nearest Neighbor algorithms. Results for AND (two agree) and OR combination (one agree) are presented. The AND combination is superior, but it does not reach the level of the neural networks using method 3 or 6 for acceptance.

Table 14 summarizes the results for combining the Neural Network (acceptance method 3) with Naive Bayes. The results show no approvement over the Neural Network method alone.

Table 13
Combination of Naive Bayes and Nearest Neighbor algorithms

	And			Or		
	F_1	R	P	F_1	R	P
Earn	0.731	0.709	0.753	0.681	0.840	0.573
Acq	0.505	0.609	0.431	0.429	0.816	0.291
Money	0.500	0.666	0.400	0.393	0.680	0.276
Grain	0.412	0.425	0.400	0.321	0.392	0.272
Crude	0.488	0.445	0.541	0.259	0.391	0.194
Trade	0.522	0.615	0.453	0.459	0.633	0.360
Int	0.396	0.481	0.337	0.409	0.378	0.447
Ship	0.418	0.348	0.500	0.351	0.364	0.339
Wheat	0.367	0.629	0.260	0.475	0.736	0.351
Corn	0.302	0.413	0.238	0.204	0.516	0.127
Avg	0.464	0.534	0.431	0.398	0.574	0.323
Wt.avg	0.570	0.614	0.548	0.501	0.710	0.402

Table 14
Combination of Neural Networks and Naive Bayes algorithms

	And			Or		
	F_1	R	P	F_1	R	P
Earn	0.789	0.780	0.799	0.747	0.850	0.666
Acq	0.531	0.544	0.519	0.536	0.601	0.483
Money	0.556	0.625	0.502	0.550	0.612	0.500
Grain	0.454	0.385	0.553	0.427	0.462	0.397
Crude	0.526	0.533	0.519	0.513	0.565	0.469
Trade	0.568	0.568	0.568	0.576	0.625	0.534
Int	0.485	0.524	0.451	0.521	0.500	0.544
Ship	0.365	0.466	0.300	0.335	0.533	0.244
Wheat	0.470	0.489	0.452	0.167	0.682	0.095
Corn	0.332	0.266	0.441	0.332	0.532	0.241
Avg	0.507	0.518	0.510	0.470	0.596	0.417
Wt.avg	0.617	0.621	0.618	0.593	0.682	0.533

Table 15 summarizes the results for combining the Neural Network (acceptance method 3) with Nearest Neighbor. The results show no improvement over the Neural Network method alone.

In Table 16, we summarize the results for combining the Neural Networks (acceptance method 3), Naive Bayes and the Nearest Neighbor algorithms. Results for OR (one agree) combination, Majority combination (two agree) and AND combination (three agree) are presented. The only mechanism to show a somewhat significant improvement is the majority combination.

Table 15
Combination of Neural Networks and Nearest Neighbor algorithms

	And			Or		
	F_1	R	P	F_1	R	P
Earn	0.797	0.792	0.802	0.762	0.770	0.755
Acq	0.505	0.736	0.385	0.531	0.544	0.519
Money	0.538	0.630	0.469	0.538	0.552	0.524
Grain	0.317	0.317	0.316	0.415	0.394	0.439
Crude	0.446	0.404	0.497	0.526	0.533	0.519
Trade	0.569	0.531	0.613	0.578	0.690	0.497
Int	0.503	0.440	0.588	0.515	0.518	0.512
Ship	0.417	0.507	0.354	0.365	0.466	0.300
Wheat	0.515	0.403	0.714	0.209	0.741	0.116
Corn	0.317	0.434	0.250	0.303	0.380	0.252
Avg	0.492	0.519	0.498	0.474	0.558	0.443
Wt.avg	0.602	0.656	0.582	0.597	0.628	0.582

5. Discussion

In this paper, we systematically attempted to evaluate the idea of classification of data by using a trained compression–decompression scheme. We did this in the context of document classification. The idea was that the compression–decompression scheme would be valid only on documents similar to the training set; hence classification of new documents can be done by seeing how well the scheme works on the document.

One of the major virtues of such a method is that it is intrinsically a one-class algorithm; i.e. no negative examples are needed in the training of the compression–decompression algorithm. While the limitation to positive examples is severe, this limitation is actual in many applications. (The one we have kept in mind was data retrieval over the internet.) Of course, we should not expect retrieval results to be as good as when full negative information is also available. (See [25] for some discussion on the relationship between one-class and two-class learning.)

We implemented the compression–decompression with a "bottleneck" version of the autoencoder neural network. We did extensive experiments to optimize the size of the neural network and the parameters of acceptance.

We discovered that performing what we called a "Hadamard product" on the data representation resulted in a substantial improvement in the results. Intuitively, this modification emphasizes more frequent words in the training set at the expense of more infrequent words. A simple Bayesian explanation for this phenomenon was also presented.

Extensive experiments were performed using different methods to determine the acceptance criteria on the filter (i.e. the "threshold"). We found two methods that yielded good results on the data examined. The first (method 3) used an analysis of changes in the F_1 measure which, however, requires some negative information (albeit *after* the training phase).

The second (method 6(b)) used only positive information and produced in many ways the best results.

To test the efficacy of our methods we did a series of comparative studies on the standard *Reuters* database using positive information variants of several algorithms (Nearest Neighbor, Rocchio, Naive Bayes and Distance Based Probability). For each of these, we performed

Table 16
Combining Neural Networks, Nearest Neighbor and Naive Bayes algorithms

	One agree (OR)			Two agree (Majority)			Three agree (AND)		
	F_1	R	P	F_1	R	P	F_1	R	P
Earn	0.681	0.840	0.573	0.731	0.709	0.753	0.727	0.709	0.745
Acq	0.518	0.623	0.444	0.531	0.544	0.519	0.517	0.573	0.471
Money	0.547	0.568	0.527	0.544	0.683	0.451	0.558	0.625	0.504
Grain	0.435	0.475	0.401	0.415	0.394	0.439	0.450	0.370	0.577
Crude	0.495	0.563	0.442	0.557	0.533	0.584	0.544	0.505	0.590
Trade	0.574	0.693	0.490	0.589	0.586	0.593	0.579	0.563	0.596
Int	0.524	0.564	0.489	0.516	0.537	0.496	0.485	0.524	0.451
Ship	0.335	0.533	0.244	0.402	0.461	0.357	0.440	0.466	0.417
Wheat	0.525	0.655	0.438	0.543	0.467	0.649	0.462	0.478	0.447
Corn	0.324	0.489	0.242	0.333	0.358	0.311	0.333	0.266	0.445
Avg	0.495	0.600	0.429	0.516	0.527	0.515	0.509	0.507	0.524
Wt.avg	0.571	0.686	0.492	0.598	0.602	0.600	0.593	0.597	0.596

similar optimization techniques on their parameters. (These optimization techniques did, in fact, require negative examples; these negative examples are not used in the "training" of the methods, but only in setting of e.g. "thresholds". To be fair, in two variants, we used the same negative examples in setting the threshold of our compression filter.)

We measured the results using standard measurements (recall, precision and F_1). The results showed that using the Hadamard representation and the neural network compression–decompression classification scheme had the best results. Moreover, it seems to be possible to choose the threshold without the use of any negative examples.

Our experiments seem to indicate that this is a robust result; thereby recommending this method (6(b)) for further work.

In some additional work, we saw that using a majority rule between three of these algorithms allows for some boosting of the results (both for recall and precision). This implies that somewhat different information is being used by the different methods.

The experimental results showed that the kernel autoassociator can provide better or comparable performance for concept learning and recognition in various domains (the Reuter dataset is not used in this study). Thus, this approach could be considered in order to improve the performance of our methods applied on textual datasets.

6. Criticism, future work and directions

There are many directions in which we feel this work could be expanded and improved.

Although we carried out extensive experiments, there are so many parameters in the various learning procedures that it was impossible to check all possibilities. Some of our methodology was heuristic and it should be possible, at least in some of the cases, to come up with a more principled approach to the choice of parameters.

For example, although our experimental results with the Hadamard representation are quite clear, there is no result which shows that this is an "optimal" representation.

Similarly, our feature selection (the dictionary of keywords) is a relatively primitive mechanism. A consequence is the number of vectors with very few non-zero values. (As noted above, we dealt with this essentially by dealing with the threshold). Perhaps a sophisticated set of features would obtain better results.

Another interesting point is that many other works use a much larger number of features [6] which, at first glance, makes our choice of 20 seem somewhat unusual. However, these other works typically use the same features for all categories, whereas our method chose the most frequent features for each category separately. Therefore, since our work involved 10 categories; a more appropriate comparison from our work is about 200 features. However, to compare we did further experiments allowing the vector sizes to increase until all words were used. (See the appendix and Fig. 4). This did increase the accuracy, but only marginally. In principle, the increased size would require more training data because of the larger network, hence such a decision is problem dependent.

We would also like to point out some additional works that have appeared since the original submission of this paper. First, the method of this paper has been applied to produce an intelligent web-browser [18]. This browser annotates the links on a page with recommendations based on a model of the user built from one-class information on the user's interests.

Second, Raskutti and Kowalczyk [25] has recently shown that one class methods can be preferable to two class methods in certain circumstances where the available information is unbalanced. The experiments there use one-class and two-class versions of SVM.

Third, we mention that Zhang et al. [35] has suggested a different kind of autoassociator (a "kernel autoassociator"). In this case an inverse function is constructed to the projection to kernel space (as in SVMs) instead of being

learned in the "bottleneck" neural network as we do in this paper.

Finally, since support vector machines have recently given very good results in document retrieval using positive and negative examples [6] and for now seems to be the most promising method in many cases. We gave a full comparative study in a companion paper [17] using a modification of the method [3] appropriate for positive information. (Roughly, it works well, but is still not quite as good as the best NN compression method.) It will be very interesting to combine the ideas of compression, Hadamard product and the SVM paradigm in future research.

Acknowledgments

This work was partially supported by *HIACS*, the Haifa Interdisciplinary Center for Advanced Computer Science. This work forms part of the doctoral thesis of the second author who was supported by a University of Haifa fellowship during his studies, and afterwards hosted by the Neurocomputation Laboratory situated in the Caesarea Rothschild Institute for Interdisciplinary Computer Science. We thank Nathalie Japkowicz for lending us some Matlab software which was used during a revision of this paper. The first author thanks Oxford University for its hospitality during his sabbatical visit.

Appendix

In this appendix, we include some additional results, as well as a more complete Bayesian justification of the Hadamard product. As an additional check, we also reran our results using Matlab Neural Network Toolbox software.

Here are the results of these checks.

(1) We reran the tests using a different break between training/testing (i.e. 75%/25%) to allow comparison with other work in the literature (e.g. ModApte) [6]. There is no significant difference between these results and our earlier results. Results are reported in Table 18. (Table 17 shows the training/testing data break-down.)

(2) We reran the results varying the number of features.
   (i) In our earlier results, we chose 20 features under the influence of [32]. However, using only positive features we do not have the possibility of "relevance" features as they did. One should note that these features are subject specific thus a direct comparison with, e.g. [13,6] is difficult. Roughly, joining all the features together we used about 200 features.
   (ii) In verifying the results using a large number of features, we see some very minor improvement continues all the way to "all features". However, it is relatively small—thus it is application dependent whether one wants to proceed beyond the small number of features we chose. Figs. 3 (F_1 values) and 4 (Accuracy results) show results using a varying number of features with a fixed number(6) of hidden neurons with the Hadamard representation.
   (iii) There is a phenomenon of a surprising level of performance using only one or two features. This has been noticed before by McCallum and Nigam [19] in this category of information retrieval (i.e. one keyword does a reasonable job of classifying articles). We ignore this in our analysis.

(3) As mentioned in Section 2, we investigated the use of "expansion" as well as "compression" in an auto-encoder as discussed in [10]. The bottom line is that expansion was not useful in this application. Fig. 5 shows typical results of different F_1 values using different numbers of hidden level neurons.

(4) Since the data set is appropriate for two-class results as well, one can compare our results with results obtained from two class methods to try to see how much is lost by not using that data. Comparisons with the NN method in this paper are not exact; however, because it is an intrinsic one-class method. Dumais et al. [6] reported results for five two-class methods Find-Similar (or Rocchio's method), Naive-Bayes, Bayes-Nets, Decision Trees and Linear-SVM. Mutual information was applied to select the most informative features; 300 keywords were chosen for SVMs and Decision Tree methods while 50 keywords were used for the other methods. They showed that the micro-averaged performance for the top 10 categories (the same ones we used with the same splits) for each implemented method were: Find-Similar 64.6%, Naive-Bayes 81.5%, Bayes-Nets 85.0%, Decision Tress 88.4% and Linear-SVM 92.0%. Joachim [13] reported results for five two-class methods: Rocchio's method, Naive-Bayes, k-NN, C4.5 decision trees and SVM (POLY and RBF). Information-gain was used for selection of the most informative features and the 1000 best were used. The micro-averaged performance over the

Table 17
Number of training/test items

Category	Dic size	Num train positive	Num test positive	Num test negative
Earn	2270	2765	1081	1701
Acq	3111	1580	703	2079
Money	1965	604	197	2585
Grain	1485	414	148	2634
Crude	1755	380	186	2596
Trade	1716	351	118	2664
Interest	1167	329	131	2651
Ship	858	187	89	2693
Wheat	831	204	72	2710
Corn	739	175	57	2725

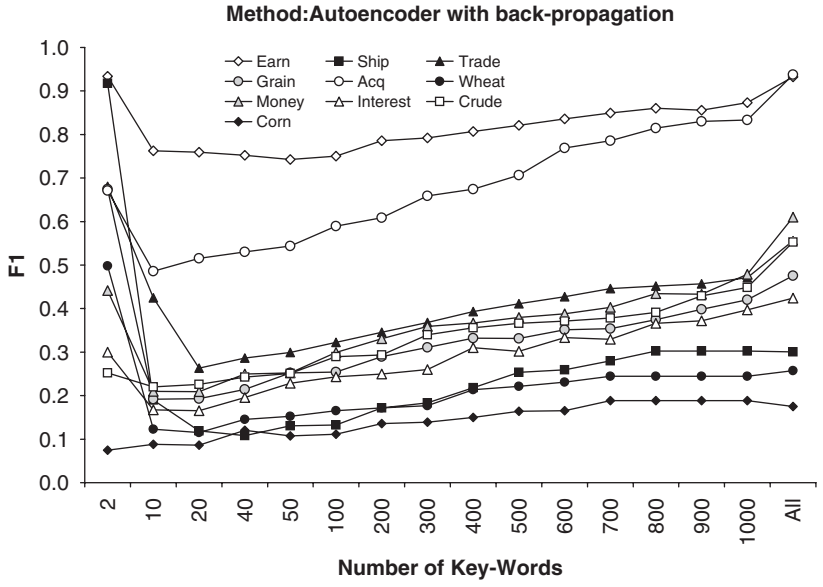

Fig. 3. F_1 results: varying the number of features with a fixed bottleneck.

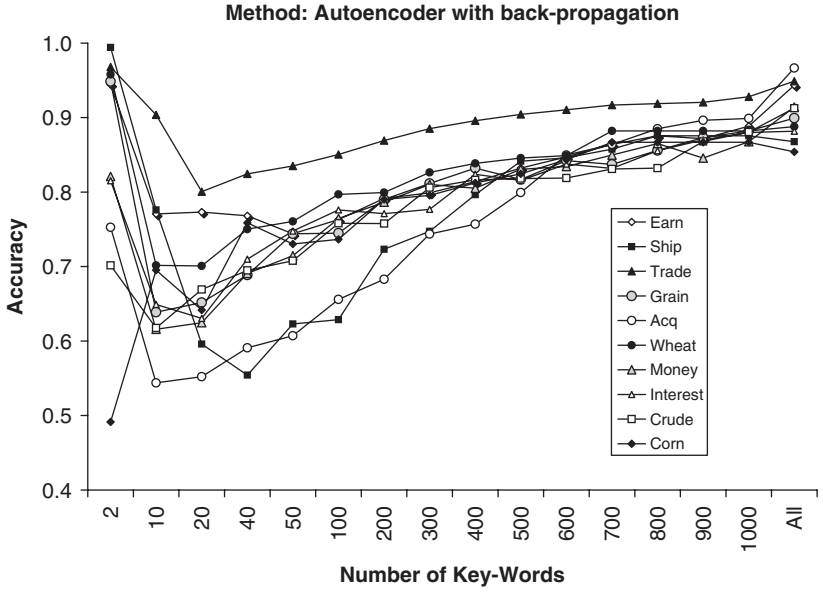

Fig. 4. Accuracy results: varying the number of features with a fixed bottleneck.

10 categories were: Naive-Bayes 72%, Rocchio 79.9%, C4.5 79.4% *k*-NN 82.3%, SVM(poly) 86.0% SVM(rbf) 86.4.

Although these results are not directly comparable with ours, recall that the best results for the NN was 61.5% with the top 20 keywords (Table 12) and 67.2% using all the keywords (Appendix, Table 18). Thus, very roughly, the loss between the one-class NN and the best results on the other two-class approaches varies between 5% and 18% for the various more "classical approaches" and between 19% and 25% for two-class SVM methods. One should bear in mind that these comparisons are over a situation where there is an excellent sampling of the second class. (See [25] for

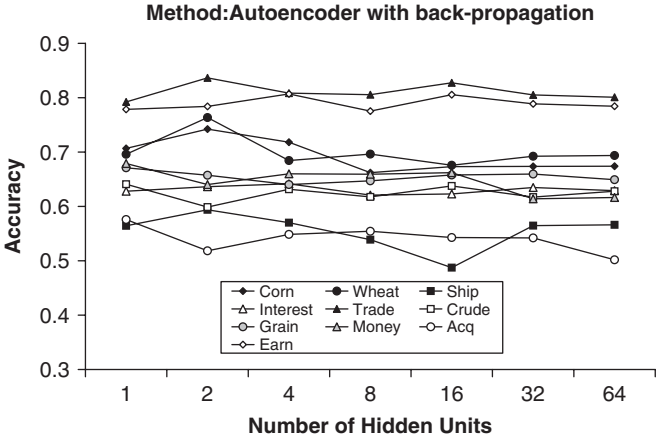

Fig. 5. Using expansions in the autoencoder.

Table 18
Comparison of Neural Networks (Hadamard representation and "All" features), and one-class SVM (Binary representation)—F_1 and accuracy with optimal number of features

	Neural Networks method 6(b) All keywords		One-class SVM Linear five keywords		One-class SVM sigmoid 50 keywords	
	F_1	ACC	F_1	ACC	F_1	ACC
Earn	0.932	0.944	0.824	0.850	0.807	0.851
Acq	0.938	0.967	0.495	0.580	0.636	0.772
Money	0.610	0.914	0.317	0.795	0.381	0.836
Grain	0.476	0.967	0.249	0.725	0.317	0.822
Crude	0.552	0.913	0.354	0.849	0.361	0.839
Trade	0.556	0.949	0.456	0.908	0.471	0.922
Int	0.424	0.882	0.378	0.891	0.326	0.861
Ship	0.300	0.868	0.357	0.938	0.178	0.791
Wheat	0.610	0.888	0.519	0.959	0.373	0.937
Corn	0.175	0.854	0.345	0.945	0.184	0.860
Avg	0.557	0.908	0.429	0.844	0.403	0.849
Wt.avg	0.672	0.908	0.512	0.844	0.506	0.849

further discussion on the comparison between one and two class learning.)

(5) Regarding the Hadamard Product, consider the following:
Let g be a document, E an interesting example.
Assume

$$P(g, E) = P(g)P(E)$$

(i.e. the independence of document and interesting example—a reasonable assumption) and assume that

$$P(g, E|w) = P(g|w)P(E|w)P(w).$$

The second assumption shows that the probabilities of g and E are not quite independent given w but have to be uniformly adjusted by the probability of w. If the a priori probability of a word is relatively uniform, then this is equivalent to the independence of g and E given w.

Under these assumptions we have

$$P(g, E, w) = P(g, E|w)P(w) = P(g|w)P(E|w)P(w)P(w).$$

Hence

$$\begin{aligned} P(w|g, E) &= P(g, E, w)/P(g, E) = \frac{P(g|w)P(E|w)P(w)P(w)}{P(g, E)} \\ &= \frac{P(g|w)P(E|w)P(w)P(w)}{P(g)P(E)} \\ &= \frac{P(w|g)P(g)P(w|E)P(E)}{P(g)P(E)} \\ &= P(w|g)P(w|E) \end{aligned}$$

which is exactly our defined Hadamard product.

References

[1] M. Balabanovic, Y. Shoham, Learning information retrieval agents: experiments with automated web browsing, in: Working Notes of AAAI Spring Symposium Series on Information Gathering from Distributed, Heterogeneous Environments, AAAI-Press, 1995.

[2] H. Bostrom, Predicate invention and learning from positive examples only, in: Proceedings of the Tenth European Conference on Machine Learning, Springer, Berlin, 1998, pp. 226–237.

[3] B. Scholkopf, J.C. Platt, J. Shawe-Taylor, A.J. Smola, R.C. Williamson, Estimating the support of a high-dimensional distribution, Technical Report, Microsoft Research, MSR-TR-99-87, November 1999.

[4] P. Datta, Characteristic concept representations, Ph.D. Thesis, University of California, Irvine, 1997.

[5] P. Domingos, M. Pazzani, Beyond independence: conditions for the optimality of the simple Bayesian classifier, in: The 13th International Conference on Machine Learning, Italy, Morgan Kaufmann, Los Altos, CA, 1996.

[6] S.T. Dumais, J. Platt, D. Heckerman, M. Sahami, Inductive learning algorithms and representations for text categorization, in: Proceedings of the Seventh International Conference on Information and Knowledge Management (CIKM'98), 1998, pp. 148–155.

[7] P. Munro, G.W. Cottrell, D. Zipser, Image compression by backpropagation: an example of extensional programming, in: N.E. Sharkey (Ed.), Advances in Cognitive Science, vol. 3, Ablex, 1988.

[8] R. Hummel, L. Manevitz, A statistical approach to the representation of uncertainty in beliefs using spread of opinions, IEEE Trans. Syst. Man Cybernetics Part A: Syst. Hum. 3 (1996) 378–384.

[9] Y. Karov, I. Dagan, D. Roth, Mistake-driven learning in text categorization, in: Advances in Classification Research, vol. 8: Proceedings of the Eighth ASIS SIG/CR Classification Research Workshop, Modford, New Jersey, 1998, pp. 59–72.

[10] N. Japkowicz, S. Hanson, M. Gluck, Nonlinear autoassociation is not equivalent to pca, Neural Comput. 42 (3) (2000) 531–545.

[11] N. Japkowicz, Supervised versus unsupervised binary-learning by feedforward neural networks, Mach. Learn. 42 (1/2) (2001) 97–122.

[12] N. Japkowicz, C. Myers, M.A. Gluck, A novelty detection approach to classification, in: IJCAI, 1995, pp. 518–523.

[13] T. Joachim, Text categorization with support vector machines: learning with many relevant features, in: Proceedings of the Tenth European Conference on Machine Learning (ECML), Springer, Berlin, 1998.

[14] T. Joachims, A probabilistic analysis of the rocchio algorithm with tfidf for text categorization, Technical Report, School of Computer Science, Carnegie Mellon University, Pittsburgh, 1996.

[15] K. Lang, Newsweeder: learning to filter news, in: Twelfth International Conference on Machine Learning, Lake Tahoe, CA, 1995.

[16] D. Lewis, Reuters-21578 text categorization test collection, ⟨http://www.research.att.com/~lewis⟩, 1997.
[17] L. Manevitz, M. Yousef, One-class SVMs for document classification, J. Mach. Learn. Res. 2 (2001) 139–154.
[18] L. Manevitz, M. Yousef, A web navigation system based on a neural network user-model trained with only positive web documents, Web Intel. Agent Syst. (WIAS) 2 (2) (2004) 137–144.
[19] A. McCallum, K. Nigam, A comparison of event models for naive bayes text classification, in: AAAI-98 Workshop on Learning for Text Categorization, Madison, WI, 1998, pp. 41–48.
[20] S. Muggleton, Learning from positive data, in: S. Muggleton (Ed.), Proceedings of the 6th International Workshop on Inductive Logic Programming, vol. 1314: Lecture Notes in Artificial Intelligence, Springer, 1996, pp. 358–376.
[21] M. Pazzani, D. Billsus, Learning and revising user profiles: the identification of interesting web sites, Mach. Learn. 27 27 (3) (1997) 313–331.
[22] M. Pazzani, J. Muramatsu, D. Billsus, Syskill & webert: identifying interesting web sites, in: AAAI Conference, 1996.
[23] M. Porter, An algorithm for suffix stripping, Program 14 (3) (1980) 130–137.
[24] C. Quek, Classification of world wide web documents, Master's Thesis, School of Computer Science Carnegie Mellon University, 1997.
[25] B. Raskutti, A. Kowalczyk, Extreme re-balancing for svms: a case study, SIGKDD Explor. Newsl. 6 (1) (2004) 60–69.
[26] G. Salton, M.J. McGill, Introduction to Modern Information Retrieval, McGraw-Hill, Berlin, 1983.
[27] H. Schwenk, The diablo classifier, Neural Comput. 10 (8) (1998) 2175–2200.
[28] B.D. Sheth. A learning approach to personalized information filtering, Master's Thesis, Massachusetts Institute of Technology, 1994.
[29] K. Swingler, Applying Neural Networks: a Practical Guide, Academic Press, New York, 1996.
[30] K. Lagus, T. Honkela, S. Kaski, T. Kohonen, Newsgroup exploration with websom method and browsing interface, Technical Report, Helsinki University of Technology, Laboratory of Computer and Information Science, 1996.
[31] C.J. van Rijsbergen, Information Retrieval, second ed., Butterworths, London, 1979.
[32] E. Weiner, J.O. Pedersen, A.S. Weignedm A neural network approach to topic spotting, in: Fourth Symposium on Document Analysis and Information Retrieval, Las Vegas, 1995, pp. 317–332.
[33] M. Yousef, Document classification using positive examples only, Ph.D. Thesis, University of Haifa, 2002.
[34] Y. Yang, J.O. Pedersen, A comparative study on feature selection in text categorization, in: D.H. Fisher (Ed.), Proceedings of ICML-97 14th International Conference on Machine Learning, Nashville, US, Morgan Kaufmann Publishers, San Francisco, US, 1997, pp. 412–420
[35] H. Zhang, W. Huang, Z. Huang, B. Zhang, A kernel autoassociator approach to pattern classification, IEEE Trans. Syst. Man Cybern. Part B: Cybernet. 35 (3) (2005) 593–606.

Larry M. Manevitz received his B.Sc. from Brooklyn College, NY and his M.Phil. and Ph.D. from Yale University of Mathematics Department in Mathematical Logic. He has done substantial work in both theoretical and applied mathematical logic (especially non-standard analysis), in the theory of combining uncertain information, and in both artificial neural networks and brain modeling. He has held positions at Hebrew University, Bar Ilan University, Oxford University; NASA Ames, University of Texas at Austin, Baruch College, CUNY, University of Maryland and University of Wisconsin. Currently he is in the Department of Computer Science, University of Haifa, and is the Director of the HIACS research laboratory and the Neurocomputation Laboratory there. He has about 60 scientific publications. Manevitz is currently in the Department of Computer Science, University of Haifa, Israel and is the Director of the HIACS research laboratory and the Neurocomputation Laboratory there.

Malik Yousef was born in Dabburia Village, Israel. In 2001, he received his Ph.D. in Computer Science and Mathematics from the Haifa University, Israel. In 2004. He joined the Showe Laboratory at the Wistar Institute in Philadelphia, USA as a Post-Doctoral Fellow. His research interests include Machine Learning and Computational Biology.