Recall Systems: Efficient Learning and Use of Category Indices

Omid Madani
Yahoo! Research
Burbank, CA 91504
madani@yahoo-inc.com

Wiley Greiner
Los Angeles Software
Santa Monica, CA 90405
w.greiner@lasoft.com

David Kempe
University of Southern California
Los Angeles, CA 90089-0781
dkempe@usc.edu

Mohammad R. Salavatipour
University of Alberta
Edmonton, AB T6G2E8
mreza@cs.ualberta.ca

Abstract

We introduce the framework of *recall systems* for efficient learning and retrieval of categories when the number of categories is large. A recall-system here is a simple feature-based intermediate filtering step which reduces the potential categories for an instance to a small manageable set. The correct categories from this set can then be determined using traditional classifiers. We present a formalization of the index learning problem and establish NP-hardness and approximation hardness. We proceed to give an efficient heuristic for learning indices, and evaluate it on several large data sets. In our experiments, the index is learned within minutes, and reduces the number of categories by several orders of magnitude, without affecting the quality of classification overall.

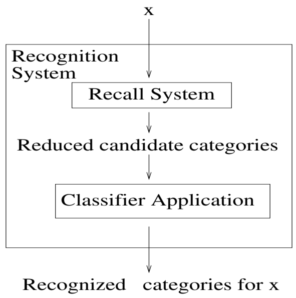
1 Introduction

A core problem in machine learning and artificial intelligence is to recognize into which of many candidate categories a given instance falls. This problem has many natural applications, including: (1) categorizing web pages into the Yahoo! topic hierarchy (http://dir.yahoo.com) [LYW$^+$05, GM98], (2) predicting words or modeling language [Goo01, EZR00], and (3) determining the visual categories for image tagging and object recognition [WLW01, FP03]. Beyond the natural classification formulation, these example applications share another crucial property: the number of training instances is very large or practically unbounded, and the number of categories can be extremely large as well. This leads to challenges in ensuring that both learning of categories and categorization of items be efficient in their usage of time and space.

In this paper, we propose a solution approach we term *recall system*. A recall system is based on introducing a layer of indirection between instances and categories, namely *features*. The number of useful features is significantly smaller than the number of instances. We use supervised learning to learn an index, *i.e.*, a mapping from features to categories, from training examples that consist of instances labeled with correct applicable features and categories. When retrieving categories for an instance, the recall system will return all the categories indexed by *at least one* of the instance's features.

As a result, a recall system will have very few false negatives, but may have more false positives. In other words, the focus is on excellent recall, while allowing for worse precision. In order to improve precision, the recall system can then be used within a *recognition system*, which applies more specific classifiers to all categories returned by the recall system, in order to narrow down the number of categories the whole system returns. Figure 1 gives a high level overview of a simple recall and recognition system.

In Sec. 2, we formally define the index learning problem, and establish both NP-hardness and approximation hardness. We then present a simple and efficient heuristic which learns an index in an online mistake-driven manner, and show its convergence in an ideal setting. Sec. 3 describes the data set we use for the experimental evaluation of our algorithm, and presents the experimental results. We test our framework both for text categorization and for word prediction [LYRL04, EZR00]. Word prediction in particular has thousands of categories and practically unbounded instances, and thus demonstrates the potential of the approach for making massive discriminative learning feasible. Text categorization is attractive as a testbed because the choice of features is generally agreed on (see, *e.g.*, [LYRL04]). In addition, much training data is now often available. We find that learning the index is relatively fast, on the order of minutes for thousands of categories and hundreds of thousands of instances. (On the other hand, standard training of all the classifiers would take on the order of hours or days.) The recall system significantly accelerates training, as it tends to reduce the number of classifiers to be updated from hundreds or thousands to tens. The accuracy achieved is similar to when the same online learners are trained on all instances. In Sec. 4 we discuss



Repeat
1. Input instance x
2. Retrieve candidate categories for x
3. Apply corresponding classifiers to x
// During learning:
4. Update the retrieval subsystem
5. Update the retrieved classifiers

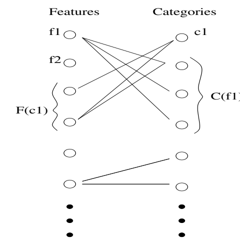


Figure 1: The basic structure of category retrieval in a recall system, and the execution cycle of a simple recognition system as two stage process. The recall system consists of steps 1 and 2 and, when learning, step 4. Right: View of an index as a bipartite graph. The edge set is learned. Not all features necessarily index (map to) a category.

related work, and conclude in Sec. 5.

2 Indexing Categories

An ideal recall system, when presented with an instance, will quickly reduce the set of possible categories to a few without loosing the right categories. Our recall system is realized by an inverted index which maps each feature to a set of zero or more categories.

We say a feature is active in an instance if it has positive value. In this paper, for learning the index (and not the classifiers), we treat features as boolean, either active or not. An instance belongs to one or more categories. Let $v[i]$ denote the value of feature i in vector v. An instance x may be viewed as a pair $\langle F_x, Y_x \rangle$, where F_x denotes the set of features active in x, and $Y_x \neq \emptyset$ the set of categories x belongs to. S is a finite set or an infinite sequence of such instances. We use $\tilde{Y}_x$ to denote a set of candidate categories calculated for instance x by a recall system.

It is helpful to view an index as a bipartite graph of features and categories (Fig. 1). There is an edge connecting feature f and category c iff f maps to c in the index. Let $C(f)$ be the set of categories that feature f maps to. Then, the recall system, on input x, retrieves the set $\tilde{Y}_x = \bigcup_{f \in F_x} C(f)$. Thus, the index implements a disjunction condition for the retrieval of each category, meaning that if a category c is indexed by features f and f', then c is retrieved whenever at least one of f or f' is active: $f \in F_x$ or $f' \in F_x$.

The execution cycle of a simple recall and recognition system at a high level is given in Fig. 1. We seek a system which on average quickly outputs an approximate set of categories $\tilde{Y}_x$ with the property that: (1) $Y_x \subseteq \tilde{Y}_x$, and (2) $|\tilde{Y}_x \setminus Y_x|$ is small. Further processing may then be performed on x and $\tilde{Y}_x$, for example sorting or ranking $\tilde{Y}_x$. In this paper, (binary) classifiers corresponding to the categories in $\tilde{Y}_x$ are applied to the instance to more precisely determine the categories of the instance. The binary classifiers may also be trained while learning the index. Note that the recall system imposes a distribution on the instances presented to the learning algorithms of each category. In order to be able to learn the classifiers within the system, they themselves have to be online, as that is the way in which instances will be presented to them. Thus, Perceptron and Winnow [Ros58, Lit88], among others, are natural choices for step 5 in Fig. 1.

2.1 Problem Formulation and Complexity

A first question is the computational complexity of the problem: are there efficient algorithms which, given any problem instance, can find an index that does well on the training set? A problem instance here is a finite set S of (training) instances, $\langle F_x, Y_x \rangle$, specified by features and true categories for each instance.

In order to evaluate the quality of a computed index, we use the following notation. Let $\Phi_-(x)$ denote the set of false negative categories on instance x, *i.e.*, the set of all true categories missed by the index on x: $\Phi_-(x) = Y_x \setminus \tilde{Y}_x$. Similarly, let $\Phi_+(x)$ denote the set of false positive categories retrieved by the index on instance x: $\Phi_+(x) = \tilde{Y}_x \setminus Y_x$. The false negative count is the total number of false negatives for all instances: $\phi_- = \sum_{x \in S} |\Phi_-(x)|$; similarly, the false positive count is $\phi_+ = \sum_{x \in S} |\Phi_+(x)|$. We define a (θ_-, θ_+)-index (on a finite training set) as an index with $\phi_- \leq \theta_-$ and $\phi_+ \leq \theta_+$. The decision problem is: given a finite set S of training instances and desired thresholds θ_- and θ_+, is there an (θ_-, θ_+)-index?

Theorem 2.1 *Given a learning problem with instance set S as well as thresholds θ_- and θ_+, let θ_+^* be the minimum fp-count in an index with fn-count at most θ_-. For any $\epsilon > 0$: (1) it is NP-hard to compute an (θ_-, θ_+)-index with $\theta_+ < \theta_+^*(1-\epsilon)\ln B$ with B being the maximum number of instances to which a feature belongs; (2) it is NP-hard to compute an (θ_-, θ_+)-index with $\theta_+ < \theta_+^*(k-1-\epsilon)$ with $k \geq 3$ being the maximum number of features in an instance.*

Proof. We give a reduction from the SET COVER problem [GJ79]. An instance $\mathcal{I}$ of SET COVER consist of a set $U = \{e_1, \ldots, e_n\}$ of elements and a set $\mathcal{S} = \{S_1, \ldots, S_m\}$ of subsets of U. The goal is to find a smallest subset $\mathcal{S}' \subseteq S$

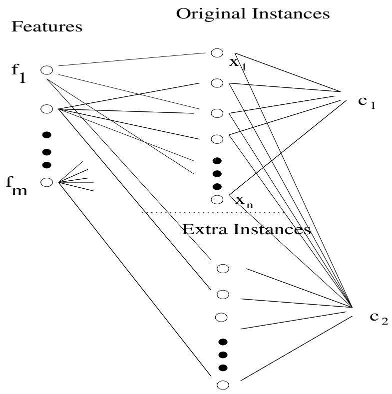


Figure 2: Reduction of minimum set cover problem to the ($0,\theta_+$)-index problem.

such that $\bigcup_{S_i \in \mathcal{S}'} = U$. Given a SET COVER instance $\mathcal{I}$, we construct an instance of the indexing problem with only two categories c_1 and c_2 such that there is a SET COVER solution of size C for $\mathcal{I}$ iff there is a $(0, \theta_+)$-index with fp-count $\theta_+ = C$. There is one feature f_i corresponding to each set $S_i \in \mathcal{S}$, for a total of m features. There is also one instance x_j for each element $e_j \in U$ $(1 \leq j \leq n)$, and x_j contains feature f_i (x_j is connected to f_i) iff the element e_j belongs to the set S_i. These instances, called the "original instances", belong to both categories c_1 and c_2. In addition, there are m "extra" instances, one for each set (or each feature). Each of these extra instances contains only the feature it corresponds to, and belongs only to category c_2 (see Fig. 2).

We will be concerned only with indices with an fn-count of 0. First, note that for any such index in this indexing problem, every feature in the index must be connected to c_2, as c_2 must be retrieved for all the instances. Now, if the SET COVER instance has a cover of size C, then the index in which only the corresponding features are connected to c_1 has an fp-count of C on the extra instances (only C of the extra instances incur a false positive cost), for a total fp-count of C. The converse can also be verified to be true: if there is an index with fp-count C, then there must exist a corresponding cover of size C.

Relaxing θ_- to be positive does not make the problem easier: one can add extra instances that cannot be covered. Relaxing θ_+ to be greater does not make the problem easier either: one can add extra categories and instances to each feature in the reduction.

It is known [Fei98, Tre01] that approximating SET COVER instances with set sizes bounded by B to within a factor of $(1-\epsilon)\ln B$ is NP-hard. Also, Dinur et al. [DGKR03] show that approximating SET COVER instances in which every element belongs to at most k sets (for $k \geq 3$) to within a factor of $k-1-\epsilon$ is NP-hard. From these two result, the theorem follows immediately. $\square$

Empirically, in high dimensional problems, the number of features is large and in fact the problems are often closer to being under-constrained. And we really seek indices that have adequate *generalization*, and not just perform well on the given training set. For a fixed index, let the *fn-rate* and the *fp-rate* be the expectations of $|\Phi_-(x)|$ and $|\Phi_+(x)|$: fn-rate $= E_{x\sim X}|\Phi_-(x)| = E_{x\sim X}|Y_x \setminus \tilde{Y}_x|$, fp-rate $= E_{x\sim X}|\Phi_+(x)| = E_{x\sim X}|\tilde{Y}_x \setminus Y_x|$, where $x \sim X$ means drawing instance x from distribution X, and E denotes the expectation. A good index has low fn and fp rates. We will see in Sec. 3 that our algorithms presented next efficiently find good indices on real data sets. A good question is whether one can characterize realistic problems adequately so that we can better explain the observations that sufficiently good indices exist and can be found quickly.

2.2 Algorithms

In light of the approximation hardness results proved above, for the rest of the paper, we focus on a fast and simple heuristic, and evaluate its performance in practice. The index is learned by the *indexer* algorithm shown in Fig. 2.2. The indexer algorithm has two parameters, a *tolerance* $\tau \geq 0$ and a *learning rate* $r > 1$. It maintains, for each category c, a sparse vector v_c of feature weights. These vectors are initialized to be all 0. In each iteration, we will have a temporary index, in which a category is indexed by those features whose weights in the category vector exceed a specified *inclusion threshold* g. That is, $c \in C(f_i)$ iff $v_c[i] > g$.

When an instance x causes a false negative or false positive, the indexer calls an update subroutine. In the case of a false negative $c \in \Phi_-(x)$, all features of the instance x are *promoted*: their vector entries $v_c[i]$ are multiplied by the learning rate r. This aggressive rule follows our intent to ensure good recall, i.e., few false negatives. False positives $c \in \Phi_+(x)$ only trigger an update if the number of false positives exceeds the the given threshold ($\Phi_+(x) > \tau$), in which case all features $f_i \in F_x$ are *demoted*, by reducing their weights for all false positive categories c by a factor of r. Afterwards, the index is updated according to the threshold g, i.e., edges are removed or inserted.

Given that the update rules are multiplicative, we need a special rule for the first promotion of a feature. Our experiments suggest that initializing a feature f_i's weight to $1/\mathrm{df}(i)$, where $\mathrm{df}(i)$ is feature i's "document frequency" (the number of instances in which the feature has been seen so far) is a significantly better choice than using unit weights. Hence, all our experiments use this initialization. The measure is related to the common tfidf weighting and its variants [LYRL04]; we are using it here not to weigh the features in the documents, but as a bias in initializing feature weights in the category vectors. The normalization by the max feature weight (step 2 in Update) guarantees that

```
Algorithm Indexer(S, τ, r, g)
 1. Begin with an empty index:
    ∀f ∈ F, C(f) ← ∅,  and ∀c ∈ C, v_c ← 0⃗
 2. For each instance x in the sample S:
  2.1 Retrieve categories:  ⋃_{f_i ∈ F_x} C(f_i)
  2.2 Promote for each false negative
      category c ∈ Φ_−(x):
        Update(x, c, r, g)
  2.3 If fp count exceeds tolerance τ,
      then demote for each false positive
      category c ∈ Φ_+(x):
        Update(x, c, 1/r, g)
  2.4 For each of the retrieved
      categories, apply their classifiers;
      update the classifiers as necessary.

Subroutine Update(x, c, r, g)
 1. For every feature f_i ∈ F_x,
      If (v_c[i] is 0, and r > 1),
         then v_c[i] ← 1/df(i),
         else v_c[i] ← v_c[i] · r
 2*. Max normalize v_c : ∀i,  v_c[i] ← v_c[i] / max_j v_c[j]
 3. Update index for c so this condition
    holds again:  c ∈ C(f_i)  iff  v_c[i] > g
```

Figure 3: The Indexer algorithm for computing an index. S is a set of training instances, $\tau > 0$ is a tolerance on the false positive count, $r > 1$ is the learning rate, and $g > 0$ is the inclusion threshold.

every category, once seen during training, is indexed by at least one feature, and may be especially beneficial for rare categories. However, in our experiments, we have not seen it impact the average recall (or the false negative rate). We see it as optional and leave it in the description of the algorithm, as in our experiments it was performed by default.

2.3 Running Time and Convergence

The update subroutine takes time $O(|x| + |v_c|)$[1]. Note that the algorithm is presented conceptually; we implement individual operations efficiently. For example, in step 1 of the indexer algorithm, we may not know the number of features or categories a priori, and so do not initialize value explicitly. Similarly, in steps 1 and 2 of the update routine, the algorithm only goes through the features that are explicitly given in F_x and v_c. The index is implemented by a doubly linked list of features, and there are pointers from features active in v_c to their entries in the index. Thus, step 3 takes time $O(|v_c|)$ as well. The indexer algorithm has been designed intentionally to be simple and fast, leaving the task of more accurate classification to the classifiers.

The indexer algorithm is guaranteed to converge to a perfect $(0, 0)$-index if one exists and tolerance is set to 0. Here, we assume the online setting, where instances are presented one after another. A feature f is "pure" for a category c, if, whenever an instance contains f, then the instance belongs to category c. Thus, a pure feature is never demoted for its category. Therefore, if all features' weights are initialized to 1, then pure features remain at 1. In fact, assuming the average number of features in an instance is ℓ, and a category has k pure features, the category is a false negative at most k times, each time a pure feature is encountered by the indexer for the first time. At every such point, the remaining possibly irrelevant features, on average $\ell - 1$, are added to the index as well, and each take $O(1)$ demotions to disappear from the index. Therefore, the number of demotions is at most $O(k\ell)$, and the total number of updates is also $O(k\ell)$. The convergence time has a dependence on both ℓ and k. While idealized, this analysis is worst-case in the following sense: it assumes that each pure feature will require $O(\ell)$ demotions of the remaining irrelevant features, regardless of the other pure features. Since we proved above (in Theorem 2.1) that the problem is NP-hard for $\theta_+ > 0$, we of course cannot expect convergence to the best possible index in those cases. However, as we will see in the next section, the performance of the algorithm in practice is quite good.

3 Experiments

In this section, we describe experiments with the indexer algorithm on several massive data sets. Our goal is to evaluate both the efficiency of the algorithm and the recall and accuracy of the index it produces. Clearly, there will be dependencies: combining the recall system with category classifiers will lead to slower learning, but higher accuracy. Also, the number of passes[2] that the indexer algorithm performs through the data will affect the recall and accuracy level, in addition to increasing the running time.

We used four large categorization data sets (Figure 4): Reuters, Ads, ODP (Open Directory Project, http://dmoz.org/), and the concatenation of six online novels by Jane Austen from Project Gutenberg (http://www.gutenberg.org/). The categories for Reuters, Ads, and ODP formed a hierarchy. Each document was labeled with all of the categories on the paths to the nodes that it was categorized under. The Reuters data is a tokenized version of the Reuters RCV1 corpus [LYRL04]. We obtained the Ads data from Yahoo! The ODP data was obtained by standard crawling and tokenizing using the Nutch search engine of the pages in the Open Directory Project. To speed up the experiments, we used only a random 330k subsample of the crawled pages. Our preprocessing (*e.g.*, unigrams and bigrams as features, stop word removal, and l2 normalization) is standard; we leave the details as well as more information on the categories in the data sets to the tech report [MG06].

[1] If a sparse vector representation is used for v_c, and the max normalization step is removed, then the update can be implemented in time $O(|x|)$.

[2] While the algorithm works in an online manner, the same instances can be presented multiple times, and this often improves the performance (like other online algorithms).

Domains	M	N	$\|\mathcal{C}\|$	l	C_{avg}
Reuters	$804,414$	$47,236$	453	75.7	3.9
Ads	2,576,118	662,280	12,827	27.3	4.2
ODP	326,338	3,422,164	70,431	331	4.9
J. Austen	906,072	94,444	13,346	11.8	1

Figure 4: Domains: M is number of instances, N is the number of features, $|\mathcal{C}|$ is the number of categories, l is the average vector length, and C_{avg} is the average number of categories per instance.

	$W^{(1)}$	$FP^{(1)}$	$FP^{(2)}$	$FN^{(1)}$	$FN^{(2)}$	$FN^{(10)}$
Reuters, $d^{(1)} = 1m$, $d^{(10)} = 1.4m$, $d^{(20)} = 1.8m$, $T^{(20)} = 0.46h$						
Train	68	37	40	0.3	0.2	0.18
Test	72	38	41	0.23	0.23	0.22
Ads, $d^{(1)} = 0.8m$, $d^{(10)} = 0.75m$, $d^{(20)} = 0.8m$, $T^{(20)} = 0.26h$						
Train	89	46	44	0.1	0.016	0.003
Test	89	47	46	0.15	0.136	0.11
ODP, $d^{(1)} = 74m$, $d^{(2)} = 56m$, $d^{(20)} = 0.43m$,$T^{(20)} = 4.15h$						
Train	237	147	59	2.38	0.38	0.004
Test	144	86	55	2.16	2.22	2.27

Figure 5: Indexer's performance. $W^{(i)}$ is average number of categories touched during retrieval in pass i, $FP^{(i)}$ and $FN^{(i)}$ denote the fp and fn-rates at pass i (eg the average fp count per instance), $d^{(i)}$ denotes time taken in pass i, and $T^{(i)}$ total time taken after finishing pass i (m =minutes, h=hours).

In order to test the generalization properties of our algorithm, we randomly selected 30% of the data to be held out for each data set. In the case of repeated runs, the same 30% was held out. In all our experiments, unless otherwise specified, the learning rate is set to 1.2, the false positive tolerance is 100, and the inclusion threshold is g 0.1. The choice of 100 for the false positive threshold is chosen as a trade-off between efficiency and recall. Our experiments suggest that so long as the learning rate r is between 1.2 and 2, neither efficiency nor recall are significantly affected by the exact value. The inclusion threshold was kept constant at 0.1 for all of our experiments. In the cases when we used classifiers in addition to the recall system, we used only default parameter values, and did not tune the classifiers. All of our experiments were run on 2.4 GHz AMD Opteron processor with 64 GB of RAM.

In the description of the experiments, each pass consists of touching each training instance once. We report on performance after both a single pass as well as several passes.

3.1 Indexer's Performance

Fig. 5 shows the indexer's performance after a selection of passes. The average number of categories "touched" during retrieval, denoted by $W = \sum_{f \in F_x} |C(f)|$, is a measure of work per instance at classification time. For example, Fig. 5 shows that on Reuters, during the first pass on the training set, on average 68 categories were touched per instance, and 37 unique categories were false positive per instance. It is possible that the fp-rate can exceed the tolerance: the algorithm only strives to bring the fp-rate down, and it may not succeed, at least in the initial passes (ODP). Note that W and the fp-rates are comparable between the training and test data: the system appears to generalize well on this aspect. We also see that W is less than twice the fp-rate. We observe that the fp-rate converges to about half the specified tolerance τ. As demotion is only applied when the false positive count exceeds τ, the maximum fp-count is close to τ, but the average fp-rate can be significantly lower.

For both the Reuters and Ads data sets, the fp-rate and W did not change much with more passes (they converge quickly). For ODP, the fp-rate decreases continuously but slowly. With more passes, the training and test fn-rates improve except for ODP, but the change is more significant over the training data. In one experiment, we removed features with frequency less than 3 from the ODP data. As a result, the test performance did not change, but the training fn-rate increased and became very close to the test fn-rate. This suggests that the indexer can indeed "memorize" infrequent features, which could hurt generalization.

To estimate the variance, we ran the indexer for 10 different random splits on the Reuters and Ads data sets. The standard deviation over fn-rates at pass 20 were 0.04 for Reuters and 0.0021 for Ads. The deviation over fp-rate and W were relatively small (less than 5).

The indexer takes the longest on ODP (over 4 hours for 20 passes), while it takes less than an hour for the other two domains. Factors such as average vector length and the noise level contribute to the difficulty of ODP. The last pass for ODP takes under a minute, while the first two passes take around an hour.

3.2 Performance When Using Classifiers

Since recall systems themselves achieve their good performance through a focus on recall (at the expense of some loss in precision), it is a natural approach to combine recall systems with a second stage of learning and applying to every instance the classifiers that correspond to the retrieved categories, in order to improve precision. Here, we compare the performance (in terms of accuracy and time) of a hybrid system against a pure classifier-based system that uses one-versus-rest training and classification [RK04]. We use the online sparse mistake-driven perceptron algorithm as the learning algorithm[3]. Since one-versus-rest learning requires training the classifiers on all instances for all the categories, only the Reuters set (with less than 500 categories) was amenable to a full comparison. On the remaining two domains, we compared the accuracy only for a sub-

[3]It begins with the 0 vector. Features are added to the perceptron with nonzero weight only in the case of a promotion (false negative).

	No	Yes
$FP^{(1)}$	0.908	0.903
$FN^{(1)}$	0.982	1.06
$FP^{(1)} + FN^{(1)}$	1.89	1.91
$FP^{(10)}$	0.982	0.878
$FN^{(10)}$	0.853	0.94
$FP^{(10)} + FN^{(10)}$	1.84	1.82
$T^{(1)}$	2.6 h	0.74 h
$T^{(10)}$	52h	16h

	$F_1^{(1)}$	$F_1^{(2)}$	$F_1^{(10)}$	$F_1^{(20)}$	$T^{(1)}$	$T^{(20)}$
Reuters (50 sample categories)						
No	0.432	0.475	0.542	0.555	0.36h	14.3h
Yes	0.421	0.464	0.531	0.524	0.05h	1.75h
Ads (76 sample categories)						
No	0.451	0.578	0.715	0.731	0.22h	18.6h
Yes	0.471	0.579	0.700	0.736	0.01h	0.5h
ODP (108 sample categories)						
No	0.032	0.07	0.14	0.17	0.34h	19h
Yes	0.059	0.10	0.14	0.15	1.3h	4.5h

Figure 6: Top: Performance (average false positive and false negative) and total time taken when training classifiers for all categories in the Reuters dataset without using the recall system (NO), and using it (YES). Bottom: Classifiers' F_1 scores with and without the recall system when trained on a sample of categories, and total times.

set of the categories.

The top of Fig. 6 shows the fp-rate and the fn-rate on the Reuters data set. We observe that the performances in terms of accuracy and recall are comparable for our recall system and the approach of learning classifiers for all categories. Notice, however, that the recall system speeds up the learning process by about a factor of three.

Next, we tracked the F_1 scores (the harmonic mean of precision and recall [LYRL04]) on a random sample of categories (Table 6). When the recall system was not used, the classifiers for those categories were trained on all training instances. When the system was used, classifiers were trained only when they were retrieved or were a false negative. The recall system (the index) was trained for all the categories. Again, we observe comparable accuracy, but substantial savings in time in both tables. The only exception is for ODP for the first pass, where the indexer alone takes 74 minutes ($d^{(1)}$). In ODP, the indexer overhead hides classifier training costs, but note that we are training classifiers for a small subset of all categories. The ODP data set yielded the lowest accuracy. This is probably due to the fact that we used simple feature extraction techniques, and web pages are noisy. Nevertheless, we see that the use of the recall system only slightly degrades accuracy on any data set.

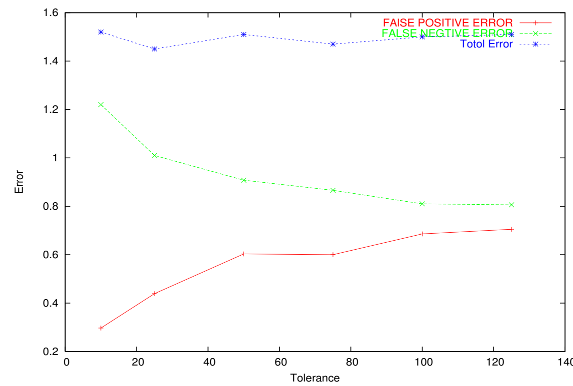


Figure 7: Accuracy of the system with classifiers (after 10 passes) as a function of tolerance, on Reuters' leaf node categories (average of two categories per instance). The total error is the sum of false positives and false negatives. The overall error remains steady, but the system is faster with lower tolerance.

3.3 Tolerance-Accuracy Trade-off

Fig. 7 shows the overall accuracy of the system when training perceptron classifiers within the system. It shows the dependency on the tolerance parameter, on a subset of the Reuters data set. As expected, the number of false negatives decreases down with increasing tolerance, while the number of false positives increases. Interestingly, the sum of the two errors remains nearly constant. The total time taken increases with higher tolerance, as expected. In this experiment, we only considered the 450 leaf categories of the data set, averaging just over two categories per instance. We have seen a similar pattern on the other data sets as tolerance is decreased from 100: the overall error remains the same or even goes down (while false negative error goes up).

3.4 Prediction in Text

Prediction tasks such as word prediction are a natural fit for classification into many categories [Goo01, EZR00, Mad06]. In these problems ample training data is available, although the labels (classes) may be viewed as noisy. These tasks find applications in speech recognition, optical handwriting recognition, and machine translation, among others.

In our experiments, we chose unigrams (single words) to predict. Any word is a candidate class (category), so each word occurrence generates an instance. For example, "I rode my bike." generates 4 instances. Features were up to 3 words in the window before and after the word, subject to sentence boundaries. Individual words in different relative positions were considered different features (*e.g.*, "rode" is a different feature in position -1 for the target class "my" and position -2 for the target class "bike").

We concatenated six of Jane Austen's novels and obtained almost 14000 unique words (Fig. 4). A random 30% of the instances were used for testing. Since we are interested in predicting words, there is only one correct category for each instance. For our experiments, we used an approach based on *ranking* all the categories retrieved by the recall system. Specifically, we considered two approaches: (1) Rank categories by the sum of the feature weights for the features that caused the category to be retrieved, or (2) Rank categories by the raw Perceptron classifier outputs (apply those classifiers that correspond to the retrieved categories). We then tracked the number of times the true category (the actual word to predict) was among the top 5 ranked categories, which we refer to as recall at 5. The results after one pass are shown in Fig. 8. Some well predicted words include: "sir", "young", and "frank". This discriminative approach is quite flexible: using stemming, parsing, or other techniques, it is possible to utilize different kinds of features [EZR00].

3.5 Misc. Comparisons

SVMs are the state of the art in text classification [LYRL04, LYW$^+$05]. Committees of online classifiers yield accuracies close to SVMs [CC06, BC03, HHK$^+$03], but carry the advantages of being online. The committee size and number of passes can be adjusted to trade off time versus accuracy. Our experience confirms their competitive accuracy: we found that often, most of the gain in accuracy is achieved already with a small number of passes and committee members. For example, we compared committees of 10 perceptrons using 10 passes against linear SVMs [KD05] on the 20 news group data set [Lan95]. We obtained an average 01-error of 0.022 for linear SVMs, and a 0.024 error for the committee. With a committee of 10 randomly initialized perceptrons for each category, the accuracy ($F_1^{(20)}$ scores) of our recall system improved from 0.53 to 0.59 (for Reuters), from 0.74 to 0.79 (for Ads) and from 0.15 to 0.19 (for ODP). Lewis *et. al.* [LYRL04] report F_1 scores for the Reuters data set comparable to our results: their macro average is roughly 0.3 for Industries and roughly 0.6 for Topic categories. They use linear SVM training and optimize the regularization parameter, but train on a smaller training set (and a chronological split).

How would simpler indexer algorithms compare? To simplify our recall system approach further, one could consider omitting category weight vectors, and instead adding edges to the index on every false-negative. Thus, one would connect the false-negative category to all features of the instance, and drop appropriate edges whenever the fp-count exceeds the tolerance, by disconnecting the false-positive category from all features of the instance. While this simple algorithm uses less memory, it can be seen to be unstable, and we have observed that the index accuracy and fn-rate suffer significantly (*e.g.*, fn-rate doubled).

4 Related Work

While batch methods for learning often provide better accuracy, past studies have pointed out the many advantages of online learning methods. These include time and memory efficiency, as well as simplicity and versatility (see, *e.g.*, [CC06, BC03, EZR00, HHK$^+$03]). For example, online training does not require instances to be stored on disk or kept in memory. For instance, in training to categorize web pages in an online fashion, each page in the training set, say from the Yahoo! topic hierarchy, can be crawled and tokenized, its vector extracted and used to update the classifiers, and then discarded. The same holds for applying discriminative learning to the problem of word prediction on large corpora. Madani [Mad06] describes and motivates prediction tasks, akin to word prediction, that need to deal with practically unbounded streams of input. In such settings, methods such as KNN [HTF01] and current muticlass versions of SVMs quickly become impractical. In addition, as alluded to by our experiments in Sec. 3.5 and shown by prior research (*e.g.*, [CC06, BC03, HHK$^+$03]), online methods often provide competitive accuracy. In many applications, simplicity and efficiency during training and classification outweigh the potential loss in accuracy. We note that many multiclass learning methods (such as multiclass Naïve Bayes) are not applicable to multilabel settings without alterations, as they assign exactly one category to each instance. Large scale discriminative learning using top-down (hierarchical) training of classifiers such as SVMs is fairly efficient both at training time and classification time, although the classification time depends on the depth of the taxonomy [LYW$^+$05]. However, this approach requires prior knowledge in the form of a taxonomy, which may not be available for some tasks, such as word prediction and object recognition. Other drawbacks include the engineering effort required to program the taxonomy structure into the training and classification architecture.

In the past, indexing has been used for efficient retrieval of explicit objects, most notably documents. In our case, categories are implicit (groupings). Grobelnik et al. also use an index of features to categories [GM98]. Their algorithm is similar to a traditional construction of an inverted index, *i.e.*, it connects a feature to all categories that contain some document with that feature. Thus, a category is the union of its positive documents. This lowers the number of categories considered per instance to less than only half of the total number of categories [GM98], which is still high. They use top-down (hierarchical) training for obtaining classifiers (*e.g.*, [LYW$^+$05]). Our learning approach makes the construction of the index dynamic and driven by an objective. To the best of our knowledge, this is the first time that index learning has been investigated.

5 Summary

In this paper, we investigated the challenging task of efficiently learning and recognizing a large number of categories. We proposed an approach termed "recall system", based on learning an index of categories. A recall system quickly reduces the number of categories under consider-

	no classifiers, b3a3	b3a3	b3a0	b0a3	b2a2	b1a1
recall at top 5	0.273	0.395	0.275	0.297	0.40	0.379

Figure 8: Word prediction: Recall at top 5 when the retrieved categories (words) are ranked, after one pass. bMaN means use M words before and N words after as features. All results except noted use classifiers' (raw) outputs, trained within the recall system. Use of classifiers and use of words appearing after and before improves ranking. Each experiment took less than 2 minutes. Tolerance was set at 20.

ation to a feasible number, at which time classifiers corresponding to the remaining categories can be applied to the instance for a more precise categorization. We presented a formalization of the problem, and established NP-hardness and approximation hardness. We then described an online index learning algorithm which performs well empirically. We showed how online learning algorithms such as perceptrons can utilize the recall system in order to efficiently learn classifiers for every category. An operational (adequately fast and accurate) system was quickly obtained via learning with relatively little programming or use of prior knowledge. Future work includes exploring alternative recall criteria, for instance learning indices that do not just retrieve categories, but also rank the retrieved categories, and alternative algorithms, possibly with provable approximation guarantees. In addition, we plan to further explore the application of our approach in various domains.

Acknowledgements

We thank Thomas Pierce, Dennis DeCoste, and Kevin Lang for pointers, discussions, or assistance, and the anonymous reviewers for their comments and suggestions. Mohammad Salavatipour's research was supported by NSERC grant No. G121210990.

References

[BC03] L. Bottou and Y. Le Cun. Large scale online learning. In *NIPS*, 2003.

[CC06] V. Carvalho and W. Cohen. Single-pass online learning: Performance, voting schemes and online feature selection. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, 2006.

[DGKR03] I. Dinur, V. Guruswami, S. Khot, and O. Regev. A new multilayered PCP and the hardness of hypergraph vertex cover. In *Proc. of the 35th ACM STOC*, 2003.

[EZR00] Y. Even-Zohar and D. Roth. A classification approach to word prediction. In *Annual meeting of the North Amercian Association of Computational Linguistics (NAACL)*, 2000.

[Fei98] U. Feige. A threshold of $\ln n$ for approximating set cover. *J. of the ACM*, 45(4):634–652, 1998.

[FP03] D. A. Forsyth and J. Ponce. *Computer Vision*. Prentice Hall, 2003.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[GM98] M. Grobelnik and D. Mladenic. Efficient text categorization. In *Text Mining Workshop at ECML*. 1998.

[Goo01] J. T. Goodman. A bit of progress in language modeling. *Computer Speech and Language*, 15(4):403–434, October 2001.

[HHK+03] E. Harrington, R. Herbrich, J. Kivinen, J. C. Platt, and R. C. Williamson. Online Bayes point machines. In *Proc. 7th Pacific-Asia Conference on Knowledge Discovery*, 2003.

[HTF01] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2001.

[KD05] S. Keerthi and D. DeCoste. A modified finite newton method for fast solution of large scale linear svms. *JMLR*, 2005.

[Lan95] K. Lang. Newsweeder: Learning to filter netnews. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 331–339, 1995.

[Lit88] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4):285–318, 1988.

[LYRL04] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004.

[LYW+05] T. Liu, Y. Yang, H. Wan, H. Zeng, Z. Chen, and W. Ma. Support vector machines classification with very large scale taxonomy. *SIGKDD Explorations*, 7, 2005.

[Mad06] O. Madani. Prediction games in infinitely rich worlds. In *Utility Based Data Mining workshop at KDD*, 2006.

[MG06] O. Madani and W. Greiner. Learning when concepts abound. Technical report, Yahoo! Research, 2006.

[RK04] R. Rifkin and A. Klautau. In defense of one-vs-all classification. *JMLR*, 5, 2004.

[Ros58] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[Tre01] L. Trevisan. Non-approximability results for optimization problems on bounded degree. In *Proc. of the 33rd ACM STOC*, 2001.

[WLW01] J. Z. Wang, J. Li, and G. Wiederhold. SIMPLIcity: Semantics-sensitive integrated matching for picture libraries. *IEEE PAMI*, 23(9):947–963, 2001.