Learning When Concepts Abound

Omid Madani
madani@yahoo-inc.com

Wiley Greiner*
w.greiner@lasoft.com

Technical Report YR-2006-101
Yahoo! Research
3333 Empire Ave
Burbank, CA 91504 USA
5/22/2006

Abstract

We introduce a framework for learning and recognition of concepts when the number of concepts is large. Central to our work is a *recall system*, a system designed for efficient high recall. The ideal recall system, given an instance, rapidly reduces the set of possibly relevant concepts of the instance to a small manageable set, without losing the actual relevant concepts. We realize the recall system by an inverted index that is learned in an online mistake driven fashion. We present a mistake bound analysis of the convergence of the algorithm in an idealized setting. We use online learners within the system to efficiently learn classifiers for all the concepts. Experiments with large text categorization corpora demonstrate the potential of our approach. In our experiments, on data sets with hundreds of thousands of instances, the index is learned very quickly, in minutes, and the learned index reduces the number of concepts to tens per instance from thousands, without affecting overall accuracy.

1 Introduction

We organize our mental world around concepts [Mur02]. Concepts are our internal representations for categories (groupings) of entities, that serve some use. Examples of concepts include hard-wired or early learned low-level concepts such as various shapes and colors ("circular","redness",..), a range of general to specific kinds such as "animal", "shape", "plant", "frog", and composite concepts such as "a pleasant song", "rocky road", and "learning when concepts abound". In this paper, we use the terms concept and category interchangeably, and a concept refers to those categories that are explicitly represented

*Research done while the author was at Yahoo! Research.

(hard-wired or learned). Questions that have been investigated extensively are how concepts may be acquired or represented, but such questions have been addressed often in isolated single concept at a time settings [Mur02]. There is little doubt that we have learned to use many concepts in our lives.

Naturally, many computational tasks can be formulated as problems that require learning and recognizing numerous concepts. In a number of existing text categorization domains, such as categorizing web pages into the Yahoo! or ODP topic hierarchies, the number of categories currently range in the hundreds of thousands. WordNet contains tens of thousands of noun, verb, and adjective categories [Mil95]. Tagging phrases in natural language text with these categories is referred to as semantic tagging [FGL99]. In the task of language modeling [Goo01, EZR00], each possible word or phrase to be predicted may be viewed as its own category, thus the number of categories can easily exceed hundreds of thousands. Similarly, visual categories useful for scene interpretation and image tagging in vision are also numerous [WLW01]. Learning techniques that can scale to myriad categories have the potential to significantly impact such tasks. We will focus on text categorization in this paper [Seb02]. Our setting is supervised learning: we assume the categories (concepts) are given and training data is available.

A fundamental question is how so many concepts are managed and rendered operational[1]. In particular, our first motivation for the present work is to address the question of efficient recognition or classification: given that a system has learned myriad concepts, when presented with an instance, how can it quickly identify the appropriate concepts from such a large set? Concepts may share many features and each may have complex representations or classifiers, for example, linear threshold functions. A brute force approach such as application of each classifier to the instance is not feasible. Perhaps there are certain cues (features) that retrieve the right concepts (or contexts) or at least narrow down the candidates significantly? We explore such a possibility in this paper. However, even if some cues or features can serve to limit the possibilities, how are these cues determined? We offer an approach based on learning.

Our second motivation goes back to the learning phase, and concerns in part efficiency, again in the setting of numerous concepts. Here we make the assumption that concepts are recognized in part by binary classifiers. In discriminative learning of binary classifiers, a learner (the learning algorithm) needs to train on negative as well positive instances. Training on all instances is not feasible when the numbers of instances and concepts are large. It would be beneficial if for each concept, the learner for that concept mostly saw the type of instances that it would be called upon to classify.

The major part of our solution for the above problems consists of a *recall system*. The defining characteristic of an ideal recall system is that on each instance, it quickly reduces the set of candidate relevant concepts of the instance to a small manageable set, for example

[1] An operational system is one that works! It satisfies some desired operationality criteria. In our case, the informal criterion is: adequately fast and accurate.

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

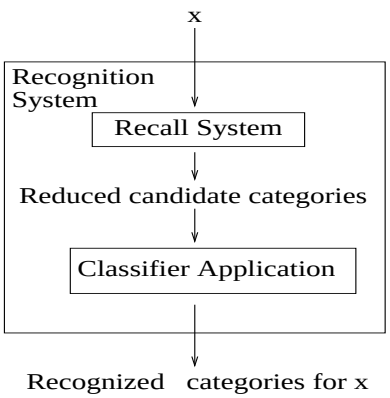

Figure 1: The execution cycle of a simple recognition and recall system and an illustration of the system during execution. The recall system consists of steps 1 and 2 and, when learning, step 4.

in the order of tens, without losing the actual relevant concepts. We realize the recall system by an inverted index mapping features to concepts. The index is constructed so that it satisfies the operationality criteria of the recall system: on presentation of an instance, ideally all the relevant concepts of the instances are retrieved, *i.e.*, the system should have high recall[2], and in addition not too many false positive concepts are retrieved. The recall system may then be used within a *recognition system*: After retrieval of candidate concepts, the recognition system applies the classifiers corresponding to the retrieved concepts to the instance in order to precisely determine the true concepts. Figure 1 gives a high level overview of a simple recall and recognition system. We expect that a recall system would have other benefits, and in particular it should facilitate the process of learning concepts in terms of other concepts. We do not pursue this possibility in the current paper.

We present an efficient algorithm to learn the index in an online mistake driven manner. We show its convergence and derive a mistake bound in an ideal setting. The classifiers can be learned within the recognition system in tandem in an online manner. This achieves efficiency during learning.

We test our framework on three large text categorization data sets: Reuters, Ads, and ODP (Section 3). Text categorization is appealing as it has properties such as large dimensionality and sparsity that is shared by many other challenging large-scale learning problems, the choice of features and vector representation for adequate performance is clear [Seb02], and ample training data is available for some data sets.

We find that learning the index is relatively fast, for example in the order of minutes for thousands of categories and hundreds of thousands of instances (while training all the classifiers takes in the order of hours or days). As intended, the recall system significantly

[2]The use of term "recall" here is related to but slightly different from its traditional use in machine learning for evaluation of a binary classifier: traditionally in machine learning, recall is computed from the point of view of the classifier. It is the proportion of actual positive instances the classifier gets right. Here, it reflects the point of view of an instance: the proportion of true concepts of an instance that is computed (*e.g.*, retrieved), averaged over instances.

accelerates training, for example it can reduce the number of classifiers retrieved to be updated to tens (from hundreds or thousands), and the accuracy achieved is similar to the scenario in which the same online learners are trained on all instances. As the recall system is so quick to train, it could be used for ranking the concepts retrieved without the need for classifiers, and while the accuracy is not as good (Section 3), for some applications such as interactive training or recommendation, such accuracy may be sufficient.

We present the recall system and the index learning algorithm in Section 2. Section 3 describes the data and presents our experiments. Section 4 discusses related work, and Section 5 concludes.

2 Recall Systems

We begin with some preliminaries, terminology and notation. More definitions and notation are given in Sections 2.1 and 2.2 and in the experiments section as needed. We utilize binary (one-vs-rest) learning. For background on machine learning as applied to text categorization see for example Sebastiani [Seb02] or Lewis *et. al.* [LYRL04]. The features in text categorization are often phrases (unigrams, bigrams,...) extracted from the document. Therefore the dimensionality is large (thousands, millions,..), while each instance (vector) is sparse (often tens to hundreds long), *i.e.*, most features have 0 value and are not explicitly represented in each instance. We say a feature is active in an instance if it has positive value. In this paper, for training the index (and not the classifiers), we treat features as boolean, either active or not. We use $|x|$ to denote the number of active features in instance x, and $x[i]$ to denote the value of feature i in vector x. Each instance x belongs to one or more (actual positive) categories y_x, and we use $\tilde{y}_x$ to denote a set of candidate categories calculated for x, say by a recall system. We denote an element in a sequence using superscript notation, *e.g.*, $F_1^{(i)}$ denotes the ith F_1 score (harmonic mean of precision and recall [Seb02]).

The execution cycle of a simple recognition system and a recall system at a high level is given in Figure 1. When an instance is input to the system, all concepts that are relevant should be retrieved efficiently. In our classification setting, relevance means that we seek concepts that are *positive* (for the instance), *i.e.*, all concepts that the instance belongs to. In general, we seek a recall system that on average on any input x quickly outputs an approximate set of categories $\tilde{y}_x$ with the property that:

1. $y_x \subseteq \tilde{y}_x$,
2. $|\tilde{y}_x - y_x|$ is small (*e.g.*, tens).

Some further processing may then be performed on x and $\tilde{y}_x$ down the stream, which will be a function of the retrieved concepts and the instance. In this work, (binary) classifiers corresponding to the concepts are applied to the instance to determine the categories

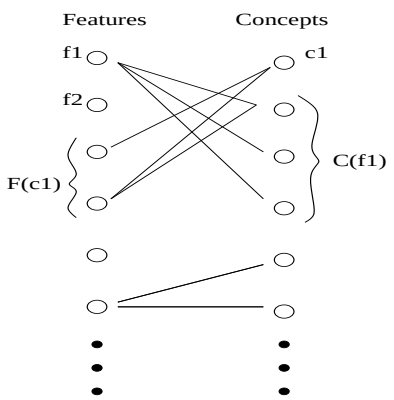

Figure 2: View of an index as a bipartite graph. The cover (the edge set) is learned. Not all features necessarily index (map to) a concept, but, if the max (l_∞) normalization is used in vector updating, any concept that has been seen before is always indexed by at least one feature.

of the instance. We also briefly report results on a method for ranking the categories in $\tilde{y}_x$. Other possibilities (not pursued here) include multiple subsequent accesses to the recall system, as needed. During training, the recall system is trained so that on average on each instance candidate concepts are retrieved efficiently, and not too many positive concepts are missed.

Optionally, binary classifiers corresponding to the retrieved concepts may also be trained within the system. Note that the recall system imposes a distribution on the instances presented to the learning algorithms of each concept. Online mistake driven[3] algorithms, such as perceptron and winnow [Ros58, Lit88], are a natural fit with the recall system. Other learning algorithms are plausible, as long as they do not necessarily require seeing all instances for training in order to perform adequately.

2.1 The Inverted Index

Our recall system is realized by an inverted index which maps each feature to a set of zero or more concepts. It is helpful to view an index as a bipartite graph of features versus concepts (Figure 2). We refer to the set of edges as a *cover*. There is an edge connecting feature f and concept c iff f maps to c in the index. Let $C(f)$ be the set of concepts that feature f maps to, and let $f(x)$ denote the set of features active (positive weight) in instance x. Then the recall system retrieves the set of concepts $\cup_{f_i \in f(x)} C(f_i)$, on input x. Efficiency requires that not only the set of irrelevant concepts that are retrieved be small, but also computing the retrieved set should be fast. We also seek an algorithm that computes a satisfactory cover efficiently.

The cover is learned by an *indexer* algorithm (see Figure 3). During learning each concept c is represented by a sparse vector of feature weights v_c. A concept is indexed by

[3]Mistake driven updates follow the general philosophy: *If it ain't broke, don't fix it!*

those features whose weight in the concept vector exceed a positive *inclusion* threshold g, $c \in C(f_i)$ iff $v_c[i] > g$. Thus the recall system implements a disjunction condition for the retrieval of each concept, meaning that if a concept c is indexed by feature f_i and f_j say, then c is retrieved whenever at least one of f_i or f_j is active in the instance.

Step 2.4 can be initiated after some time has passed in learning the cover (so that retrieval is faster and more stable).

2.2 Learning to Index

The indexer algorithm is shown in (Figure 3). It takes two nonnegative parameters, a *tolerance* τ, a *learning rate* r, where $\tau \geq 0, r > 1$. It calls an update subroutine, in this paper the Max_Norm_Update routine, which also uses an inclusion threshold g, where $g > 0$. We begin with the 0 vector for each concept and an empty index. We say a concept is a *false negative* if it is a (actual) positive concept (for the given instance) but is not retrieved. A *false positive* is a retrieved concept that is not positive (for the instance). *fp-count* and *fn-count* respectively refer to the number of false positives and false negatives retrieved for an instance. A mistake means either a false negative occurs, leading to a *promotion* update, or the fp-count exceeds threshold τ, leading to *demotion* updates. In either case one or more concept vectors and the index are updated using the learning rate to promote (increase) weights or demote (decrease). We update for every false negative due to our emphasis on recall, but tolerate relatively small numbers of positives.

The update subroutine takes[4] $O(|x| + |v_c|)$ time. In preliminary experiments, we have found that initializing a feature's weight by $1/df$, where df is its "document frequency" (that is number of instances in which the feature has been seen so far) helped significantly accelerate convergence over initializing to unit weights, for example 5 times faster for the initial pass of indexer over Reuters data, although the difference in subsequent passes drops. We use $1/df$ initialization. This measure is related to the common tfidf weighting and variants [Seb02], but we are using it here not to weigh the features in the documents, but as a bias in weighting of feature weights in the concept vectors.

Note that the operations in the figure are presented conceptually, and are implemented efficiently. For example, in step 1 of the indexer algorithm, we need not know the number of features or concepts apriori, and so do not initialize explicitly. Similarly, in steps 1 and 2 in 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 pointer from features active in v_c to their entries in the index. Thus step 3 takes $O(|v_c|)$ as well.

[4] If a hash is used for v_c, and the max normalization step is removed, then the update can take $O(|x|)$.

Algorithm Indexer(S, τ, r)

1. Begin with an empty index:
   $\forall f \in \mathcal{F}, C(f) \leftarrow \phi$, and $\forall c \in \mathcal{C}, v_c \leftarrow \vec{0}$
2. For each instance x in training sample S:
   2.1 Retrieve concepts: $\cup_{f_i \in f(x)} C(f_i)$
   2.2 Promote for each false negative concept c:
   Max_Norm_Update(x, c, r)
   2.3 If fp count is greater than tolerance τ,
   then demote for each fp concept c:
   Max_Norm_Update(x, c, r^{-1})
   2.4 For each of the retrieved concepts apply
   their classifiers and update as necessary.

Subroutine Max_Norm_Update(x, c, r)

1. For every feature $f_i \in f(x)$,
   If ($v_c[i]$ is 0, and $r > 1$),
   then $v_c[i] \leftarrow \frac{1.0}{df(i)}$,
   else $v_c[i] \leftarrow v_c[i] * r$
2. Max normalize $v_c : \forall f_i$, $v_c[i] \leftarrow \frac{v_c[i]}{\max_j v_c[j]}$
3. Update index for c so this condition
   holds again: $c \in C(f_i)$ iff $v_c[i] > g$

Figure 3: Training the recall and recognition systems, and in particular the indexer algorithm for learning the cover (excluding step 2.4). S is a set or source of training instances, τ is a tolerance on the false positive count, $\tau > 0$, r is the learning rate, $r > 1$, and g is the inclusion threshold, $g > 0$. See Sections 2.1 and 2.2 for details.

2.3 Analysis

We analyze a simplified case first. This provides some intuition on the workings of the learning algorithm and why it can be effective. We then describe more relaxed performance criteria but leave establishing convergence under such criteria for future work. Our experiments provide evidence on the efficacy of the indexer.

A feature f is "pure" for a concept c, if whenever an instance contains f, then the instance belongs to concept c. A pure feature is never punished (demoted) for its concept, as by assumption, f does not appear in an instance that does not belong to c. Therefore, if all features' weights are initialized at 1.0, then pure features remain at 1.0. In fact, assuming the average number of features in an instance is say l, and a concept requires k pure features, the number of times the concept is a false negative is at most k, each time an instance that has a pure feature is shown to the indexer for the first time. At every such point, the remaining possibly irrelevant features, on average $l - 1$ many be added to the index as well, and each take $O(1)$ demotions to go out of the index. Therefore the number of times the concept is a false positive (demotions) is at most $O(kl)$, and the total number of updates (mistakes) is also $O(kl)$. As might be expected, we see convergence has a dependence on both l and k. In practice, most features are not pure, and concepts are not effectively covered by pure features only. That is why we allow for a positive tolerance. On the other hand, this analysis is worst-case in the following sense: it assumes that each pure feature will require its $O(l)$ demotion of the remaining irrelevant features, exclusive of the other pure features. We will see in the experiments section that the number of updates per concept tends to be significantly lower.

Under a distribution X on the instances, using a fixed cover results in some average per-

formance of the system. Define the *fn-rate* (false-negative rate) of a cover as the expected fn-count, *i.e.*, number of categories missed per instance, when using that cover. Similarly, *fp-rate* denotes the expected fp-count:

$$\begin{aligned} \text{fn-rate} &= E_{x \sim X} |y_x - \tilde{y}_x| \\ \text{fp-rate} &= E_{x \sim X} |\tilde{y}_x - y_x|, \end{aligned}$$

where $x \sim X$ means drawing instance x from distribution X, and E denotes the expectation. An (fn, fp)-cover is one whose fp and fn-rates are no greater than fp and fn respectively.

The analysis above implies convergence of the indexer to a $(0, 0)$-cover when such a cover (a pure and complete cover) exists and we set tolerance to 0. Given some (fn, fp)-cover exists, does the max-norm indexer (or another algorithm) efficiently converge to such a cover when tolerance is set to say fp?

2.4 Discussion

We made a number of choices in developing the indexer algorithm. We briefly discuss some of these choices and possible alternatives. The max (l_∞) normalization could be removed, and in that case, we wouldn't have a guarantee that once a concept is seen, it is indexed (by at least one feature), which could potentially degrade recall (increase fn-rate). Our empirical experiments have not shown significant change in overall fn-rate when we remove the normalization step, but a few small concepts do fall out of index ("forgotten"?), *i.e.*, no feature maps to them.

Replacing the multiplicative updates with additive ones may not affect performance, but multiplicative updates have the advantage of facilitating normalization. In place of max normalization, we could use other normalization and in particular l_1 or "sum" (similar to the Winnow algorithm). In this case, we could further aggregate weights when features index the same concept, signifying evidence of relevance (and retrieved concepts would be ranked). In fact we began with such an approach. However, the choice of which feature to concept pointers to keep in index (thresholds, etc) was not clear, and we opted for the simpler learning a disjunction alternative for starters. Still the feature weights even in our current method may be useful in ranking concepts during retrieval at the cost of the additional overhead. In this paper, the edges are unweighted (we do not use the feature weights for that purpose). We report on a one ranking experiment in the next section. We plan to further explore ranking algorithms and ranking performance. There are many other alternatives that may enhance performance including different promotion and demotion policies, changing the tolerance with time, and batch rather than online indexer algorithms.

We expected that a simple indexer and a simple index (cover), which for example do not take feature interactions into account, would be effective. The job of the recall system is only to reduce the candidate set to a small manageable size. We leave the responsibility

of precise classification, which may take into account feature interactions, to the retrieved classifiers. Linear classifiers such as naive bayes, perceptron and linear SVMs, which do not take feature interactions into account, have been found to be quite effective in a number of learning tasks [Seb02, RYA95, EZR00].

A number of basic questions remain with our simple algorithm: How should parameters, in particular tolerance, be picked? We expected that tolerance would be a slow increasing function of number of concepts, as we did not expect features to index many concepts, on average. Existence of a cover and convergence speed depend on factors such as learnability of the concept space, average instance length, and of course the indexer and retrieval algorithms used. Still, how effective would learning simple disjunctions be? Is the overhead with constructing and using the index sufficiently low? Does a reasonable tolerance (in say tens or hundreds) yield satisfactory recall for thousands of categories? The next section sheds light on these questions.

3 Experiments

The recall system improves in performance over time. Performance includes both efficiency measures such as speed and memory requirements of the recall system, as well as measures of accuracy, in our case this includes fp-rate and fn-rate (Section 2.3). Accuracy and efficiency measures are not unrelated. For example, the higher the fp-rate, the slower the overall concept recognition time in general. On the other hand, lowering fp-rate by reducing tolerance tends to increase the fn-rate.

A first step is to determine the effectiveness of the system on its own training data. Given satisfactory performance, a next question is whether the system generalizes well, *i.e.*, whether it has satisfactory performance on unseen (test) data (Section 3.2). Another important question is how classifiers trained within the system compare (in accuracy and timing) to classifiers without the system (Section 3.4). Finally, some system aspects such as rates of convergence and size statistics are reported in Section 3.6.

The same random 30% of the data was held out in all cases, for each domain. In all our experiments here, unless otherwise specified, the learning rate is set to 1.2, tolerance at 100, and inclusion threshold at 0.1. Why did we choose such parameter values? We chose a tolerance of 100 as we had conjectured that recall of concept sizes in the order of 10s would be adequate both for efficiency as well as recall. Runs of the indexer algorithm on the Reuters and Ads data sets showed that often the average number of concepts retrieved would be around half the tolerance. The performance on the test for the indexer was similar when we used learning rate of around 2 or otherwise. Just as in perceptron learning, we expect that as long as the factor r is not too extreme (*e.g.*, significantly greater than 2 or less than 1.01), recall and convergence speed will both be adequate. Factors in the range of say 1.2 to 2.0 (with inclusion threshold of 0.1, and our choice of initialization) would allow for a few promotions or demotions before an edge makes it into the index or is dropped.

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

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

We never changed the choice of inclusion threshold (in any of our experiments, reported or not). Finally, in all experiments on overall classification performance, *i.e.*, when we used classifiers in conjunction with the recall system (Section 3.4), we have only used the default parameter values (*i.e.*, we did not tune).

Experiments were run on 2.4 ghz AMD Opteron processor with 64 gigs of RAM. A pass constitutes touching each instance once, whether in the train or test partition. We report on many statistics, and some notation is introduced in the figures as needed.

3.1 Data Sets

We report experiments on 3 large text categorization data sets: Reuters, Ads, and ODP (Figure 4). In all three, the categories formed a hierarchy. We would assign to a document all the labels corresponding to the categories on the path to the node(s) that it was categorized under.

The Reuters data is a tokenized version of the Reuters RCV1 corpus [LYRL04, RSW02]. The vectors are already processed (*e.g.*, stop words removed, cosine normalized). We used the industry (365 categories) and topic categories and ignored the geographical categories [5]. This yielded a hierarchy with depth 3, where the first level consists of the two root categories: industry and topic. As the average number of labels per document is close to 4 instead of 3 (Table 4), it appears that many documents are multilabelled. To get an idea of how many categories with some reasonable amount of labeled data for minimal learning were in the data set: The number of categories with more than 20 and 50 instances was 431 and 408 respectively.

We obtained the Ads data from Yahoo! Basic processing (stop word removal, l2 normalization) was performed. The Ads have 22 categories in the first level, and average category depth is around 4. The number of categories with more than 20 and 50 instances was 10340 and 7787 respectively.

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, specially to be able to compare to standard (all instances) training, we used only a random 330k subsample of the crawled pages. This data set turned out to yield the lowest performance in

[5] We we were concerned that they may not be suited for bag-of-words categorization and could skew our classifier accuracy results (either too high or too low).

	$W^{(1)}$	$FP^{(1)}$	$FP^{(2)}$	$FN^{(1)}$	$FN^{(2)}$	$FN^{(10)}$
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

Table 1: Indexer's performance. $W^{(i)}$ is average number of concepts touched during retrieval in pass i, fp and fn denote the fp and fn-rates, $d^{(i)}$ denotes time taken in pass i, and $T^{(i)}$ total time taken after finishing pass i (m =minutes, h=hours).

terms of accuracy. We expect performance should be improved significantly by better feature extraction: we did not alter the feature extraction in Nutch. The number of categories with more than 10, 20, and 50 instances was 13367, 7325, and 2981 respectively.

3.2 Indexer's Performance

Table 1 shows indexer's performance after a selection of passes. The average (per instance) number of concepts "touched" during retrieval, denoted W, $\sum_{f_i \in f(x)} |C(f_i)|$, is a measure of work per instance at classification time. First, it is important to note that W and fp-rates on train and test are comparable. Thus in this respect, the system generalizes well. We also see that W is not much higher than the fp-rate (less than twice). We observe that the fp-rate converges to less than the tolerance, around a half or lower. As demotion is only applied when the false positive count exceeds tolerance, the maximum fp-count may converges to a number around the tolerance, while the average fp-rate of the learned index may be significantly below tolerance.

For both Reuters and Ads, fp-rate and W did not change much with more passes (perhaps a change of within 10%). 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, and the test performance did not change, but the training fn-rate increased and became very close to the test. This suggests that the indexer can indeed "memorize" infrequent features which could hurt generalization. We think that the relatively bad performance measures on the ODP is due to high label noise as well as our naive feature extraction. The performance of classifiers (reported next) were also the lowest on this data set.

To get an idea of variance, we ran the indexer for 10 different random splits on Reuters

and Ads, and 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. We expect 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.

3.3 A Look at the Number of Updates

In the analysis Section 2.3 we obtained $O(kl)$ updates for a category with k pure features. That analysis made the worst-case assumption that the number of updates required for each of the k features were independent, but made the optimistic assumption that the features are pure. The features are not pure in the data sets, and many may be redundant, but calculation of the actual number of updates per category is useful.

In a few runs on a subset containing 200k instances of Reuters, on average each concept was indexed by 300 features, and we measured the number of updates per category. Even if each feature required one update (demotion or promotion), the bound predicts roughly $300 \times 75 \approx 22k$ updates for a concept (75 is the average instance length), while the total number of updates per concept after passes 1 is just under 900 and the total goes up to only $1.8k$ after 5 passes (the performance of the learned index converges in a few passes).

In another experiment, we initialized the features with weight 1 when first introduced during promotion (instead of using the default $\frac{1}{df}$), as the theoretical analysis assumed unit initialization. We used a learning rate of 2. This time concepts were indexed by just over 400 features in the learned index, and again even if each feature required 1 update, the bound predicts roughly $400 \times 75 \approx 30k$ updates, while the total number of updates per concept after passes 1 was just under $4k$ and the total goes to $6k$ after pass 5 is finished. We also see that initializing by $1/df$ tends to speed up convergence (and the performance of the learned index in terms of the fn-rate tends to be better).

3.4 Performance When Using Classifiers

With the addition of classifier learning we expect to improve the precision of the system, while we wish to preserve a satisfactory recall level. Here, we compare performances, with and without using the recall system, when using the online sparse mistake-driven perceptron algorithm[6]. Due to time constraints, we could train on all classifiers for all categories only for Reuters. Table 2 shows the fp-rate and the fn-rate after certain passes. To test training of classifiers on all instances, we simply set the tolerance to 500, which leads to negligible fn-rate due to the indexer (less than .001). We observe that the performances

[6]Begins with the 0 vector and features are added to the perceptron with nonzero weight only in the case of a promotion (false negative).

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

Table 2: Performance and total time taken when using the system (Yes), and without (No), on Reuters, when training for all concepts.

	$F_1^{(1)}$	$F_1^{(2)}$	$F_1^{(10)}$	$F_1^{(20)}$	$T^{(1)}$	$T^{(20)}$
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

Table 3: Classifiers' F_1 scores with and without the recall system on a sample of concepts, and the total times taken.

are comparable. As might be expected, when using the system the fn-rate is higher than without. Using the system leads to more than 3 times the speed up.

Next, we tracked the F_1 scores on a random sample of categories (Table 3), *i.e.*, only classifiers for the selected concepts were trained. The classifiers were trained on all training instances (recall system not used), or only when they were retrieved or were a false negative (using the system). Again, we see comparable performances. With committees of perceptrons [HHK$^+$03], accuracy often improves and can approach accuracy of properly regularized linear SVMs. With 10 perceptrons, accuracy improved as expected: $F_1^{(20)}$ scores (when using the system) for Reuters, Ads, and ODP improved respectively to 0.59 and 0.79 and $.19$.

We see substantial savings in time in both tables. The only exception is for ODP for pass 1, where the indexer alone takes 74 minutes ($d^{(1)}$). We also observe that, especially when not using the system, duration for pass 1 is significantly less than average duration, even though one expects that the number of updates (mistakes) should decrease over time. This is due to the fact that the perceptron vectors increase in length over time, the biggest difference in average length being from pass 1 to 2. This slows down the dot product performed for every classification. Therefore, we have observed the biggest increase in duration is between passes 1 and 2. When using the system, this effect is still seen but to a lesser extent in the Reuters and Ads domains, due to overhead of learning the indexer which is highest in the first pass. In ODP, the indexer overhead hides classifier training costs (but note that this is a small subset of all classifiers).

3.5 Ranking Performance

We report briefly on a ranking experiment: as the concepts are retrieved, we can sum the number of features that retrieve each concept for a given instance. For example, one concepts may be retrieved due to 3 features in the instance, and another concept may be retrieved by one. We can then rank the retrieved concepts by this total count of retrieving features. We measure the average recall per instance at top k concepts retrieved. This simple ranking by count gives an average recall at top 5 of about 0.57 on Retuers, while if we use classifiers' raw (uncalibrated) scores for ranking, where the classifiers were trained using the recall system, the average recall goes up to ≈ 0.77. At top 15 ranked concepts, the recall score go up to ≈ 0.77 and ≈ 0.87 respecitvely. A similar difference is seen for the Ads data set. The quick training time of the recall system may be attractive for tasks in which high precision is not very important.

3.6 Other Statistics

Figure 5 shows that convergence to basically a steady state is fast and occurs within the first few 1000s of instances in case of Reuters and Ads. For these graphs, we tracked limited history averaging of a measure of interest, on training data, in an online manner, as follows: For a measure m, let $m_o^{(i)} = \lambda m^{(i)} + (1-\lambda) m_o^{(i-1)}$, where $m^{(i)}$ is the value of the measure at the ith instance seen (*e.g.*, the fp-count for the ith instance). The weighting factor λ was set to 0.001. Any measure could be monitored (index size, etc). We also examined onlines measure of promotion rate and demotion rates (*i.e.*, average number of promotions or demotions per instance) and they often converge quickly to their eventual range as well. Demotion rate goes down to below .01. However "thrashing" behavior, *i.e.*, demotion-rate remaining relatively high and each pass taking a long time, can occur if tolerance is set relatively too low or the problem is hard (when concepts are not easily separable by disjunctions). This slow-down occurred in the case of ODP (even at tolerance $\tau = 100$) when we dropped features with frequency less than 3. Dropping such infrequent features did not alter behavior for Reuters or Ads. Changing the tolerance, possibly as a function of observed behavior, may lead to a better algorithms.

The fn-rate (on the held-out) as a function of tolerance is shown in Figure 6, after passes 1 and 20. As expected, fn-rate goes down with increasing tolerance. Furthermore, more passes have an effect when tolerance is reasonable (*e.g.*, greater than 30 for Ads and 100 for Reuters). For very small tolerance values, we get very high fn-rates, but note that they are not the maximum possible. For example, the maximum possible fn-rate is 3.9 in case of Reuters (*i.e.*, avg number of categories per instance). This implies that there exist features that approximate pure features.

Table 4 gives a number of size related statistics of the cover learned after 20 passes. Example features with high out-degree in Reuters were "woodmark" (10 many), "prft" (stem of profit, 64), and "shr" (share, 59). Example categories indexed by "woodmark" are

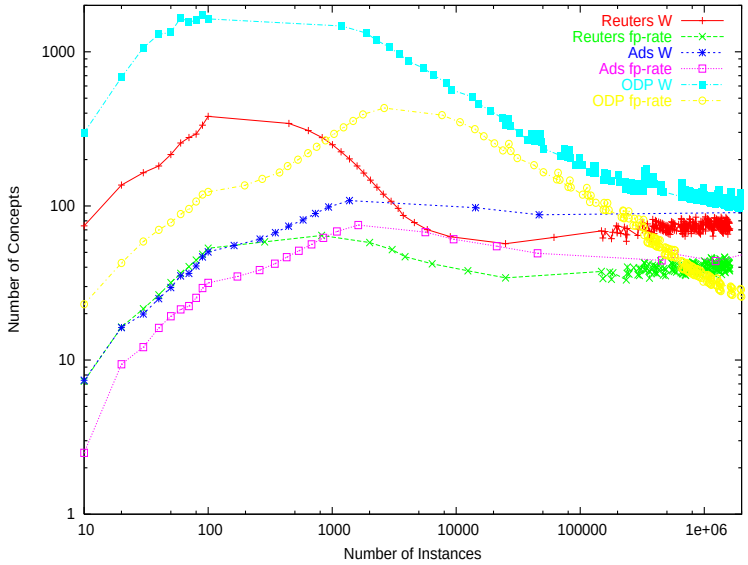

Figure 5: The rise and fall of online measures of work (W is the number of concepts touched on average) and false positive rate (number of fp concepts retrieved on average), measured on the training data, as a function of the number of training instances seen (on as a log-log plot). On the Reuters and Ads data sets the convergence is quick (within pass 1), while on ODP it is somewhat slower.

	$\|E\|$	$\|F(c)\|_{avg}$	$\|\mathcal{F}\|$	$\|C(f)\|_{max}$
Reuters	116992	258	27450	64
Ads	523639	42	173859	40
ODP	8.9×10^6	126	2.5×10^6	35

Table 4: Size statistics for the index after 20 passes: $|E|$ is the cover size (number of edges), $|F(c)|_{avg}$ is the average out-degree for a concept in the cover, $|\mathcal{F}|$ is the number feature indexing some concept (positive out-degree), and $|C(f)|_{max}$ is the maximum out-degree over features.

"WOODEN FURNITURE MEASURING" (and other furniture related), "PRECISION INSTRUMENTS" and "ELECTRONIC ACTIVE COMPONENTS" (and other equipment/component related categories).

4 Related Work

Related work includes psychology of concepts, fast recognition methods, existing candidate learning methods, online computations, perceptual and speed up learning, black-board systems, association lists and associative memory, and models of (aspects of) the brain and mind.

Cognitive psychology has stressed the importance of concepts and has focused on questions such as the nature of concepts (representation) as well as how they might be acquired

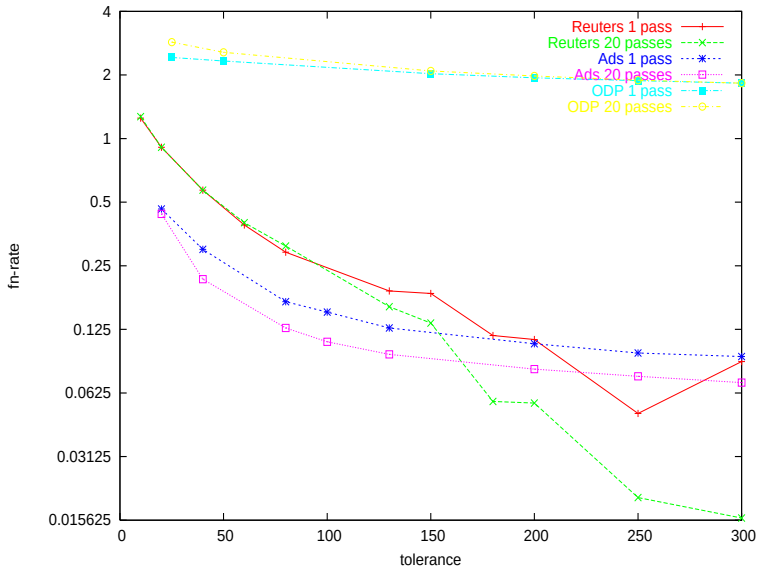

Figure 6: False negative rates on held out (y-axis) versus tolerance.

[Mur02]. Three main (representation) theories are the classical theory (logical representations), the exemplar theory (akin to nearest neighbors), and the prototype theory (akin to linear representations). The theories are far from complete, but it is believed by many that the prototype theory is the most effective overall in explaining the phenomena studied [Mur02]. It is in generally agreed that humans make use of numerous concepts. Mechanisms for managing concepts and rendering them operational are worthy of further research.

When categorizing into categories that form a hierarchy or a taxonomy, for example living things or artifacts, humans can often categorize objects the fastest into certain so-called *preferred or basic-level* categories [RMG+76, Mur02]. These categories are often not very generic or very specific. For example, categorization as a "car" may be faster than categorization into the more general class "vehicle" or the more specific class such as the make of the car (*e.g.*, "benz"). This observation is explained perhaps by the need for rapid and adequate categorization for day to day activity of a person and puts doubt on other mechansims such as hierarchical categorization methods, *i.e.*, generic to specific (see below), as a way that humans perform categorization. Many experiments on humans and animals show that recognition of visual objects is performed with ease and much speed [TFM22, GSK05]. The number of processing stages may be surprisingly small (*e.g.*, a handful) [TFM22]. Detecting the presence of an object and determining its category (*i.e.*, categorization at a preferred or basic level) appear to take the same time and may share the same processing components [GSK05]. The field of perceptual learning is also concerned in part with learning to recognize or categorize better and in particular faster [Kel02]. We think the basic philosophy behind our approach, *i.e.*, using online learning techniques to efficiently learn operational (fast and accurate) systems, to be a promising avenue for ad-

dressing the many challenges raised by the above findings.

Concepts can serve as features for learning other concepts. In fact, a major motivation for developing the recall system is to facilitate such uses. In turn, raw or low level features are simple concepts (*e.g.*, color in vision, presence of a word in text), and are relatively cheap to determine. Learning when the number of features is large has been investigated and has enjoyed many successes (*e.g.*, [Ros58, Lit88, RYA95, EZR00]). Our work adds another dimension to the large-scale learning problem: myriad concepts. It addresses fast classification and learning when there is a large number of relatively costly to detect and higher level concepts.

For fast classification, finding the relevant concepts may be cast as a problem of search for nearest points, where similarity is computed with respect to an instance at classification time. Perhaps this approach is most directly applicable in the setting where the classifiers are themselves instances, *i.e.*, nearest neighbor classification methods [HTF01]. There are a number of data structure and algorithms for fast search, including trees such as kd-trees and metric trees, algorithms based on locality sensitive-hashing, and inverted indices. Our experience with various tree based algorithms has been that they do not achieve significant speed up in very high dimensional spaces. Locality-sensitive hashing methods may work sufficiently well for approximate search [IM98, RPH05]. Note that our proposal decouples the particular representation of the classifiers from fast retrieval. Furthermore, the number of concepts, while large, can be significantly smaller than the number of instances.

The inverted index has been an invaluable data structure for efficient retrieval of documents, as well as other objects, and used even in text categorization [GM98]. Grobelnik and Meladenic use an index of feature to concepts [GM98], though their algorithm is similar to a traditional construction of an inverted index, *i.e.*, it connects a feature to all concepts that contain some document with such feature (so a concept is the union of its positive documents). The authors report that their index lowers the number of categories considered per instance to less than half of the total number of categories. They use traditional conditonal or hierarchical training for obtaining classifiers. Our robust learning approach makes the construction of index and its use much more targeted and dynamic than past uses. Classifiers may be viewed as (short) programs. Thus we are extending the use of the inverted index to efficient learning and retrieval of appropriate programs.

Our dynamic index may also be viewed as a self-organizing data structure [AW98], or an instance of online computations [CA98]. Algorithm design and analysis techniques from those areas may lead to improved methods or analyses.

Candidate methods for learning efficiently under numerous concepts include multi-class naive bayes and nearest neighbors [HTF01], and learning generative models (*e.g.*, [WLW01]): traditional nearest neighbors does not require a training phase, naive bayes requires just a single pass, and generative approaches may require only the positive instances for each concept. Naive bayes and nearest neighbors (unaltered) often suffer from poor generalization, and generative models require good models of the domain. Efficient clas-

sification remains an issue with all these methods. There are also a number of multiclass learning methods[7] [DB95, WW99], but scalability to myriad categories during learning is problematic. Rifkin and Klautu present evidence that the conceptually simple and computationally friednly one-vs-rest training of binary classifiers (the approach taken here), appears to be as good as the other more sophisticated multi-class approaches, as long as the base classifiers are properly regularized discriminative learners such as SVMs [RK04].

Learning to classify into a hierarchy (taxonomy) by conditionally training of (binary) classifiers for each category is fairly simple and, depending on the given hierarchy, can be scalable and effective (*e.g.*, [DC00, CBGTZ04]). We expect that a recall system allows ultimately for significantly more flexibility, as for instance, a hierarchy may not be applicable or available or may be unnecessarily constraining. For example, not all the semantic categories in WordNet are necessary for common semantic tagging needs, and accuracy at relatively deep categories may suffer if classifiers on the path make false negative mistakes. We hope to compare the two approaches in the future. Coarse to fine classification, applied for example to detecting faces in various poses, is an approach that shares some similarity to the generic to specific manner of hierarchical categorization [GG06]. In this setting a hierarchy is not given, and the tests to be performed, *i.e.*, the choice of classifiers to partition the data at a given stage, are computed using clustering procedures (or hand calculated), and may not be efficient when dealing with thousands of categories.

Valiant describes *random access* tasks [Val94], cognitive tasks such as memorization and inductive learning, that as he explains can require access to arbitrary parts of the neocortex, and therefore would especially strain his model of the connectivity constraints within the neocortex. The fast recognition problems we raised can also be viewed as requiring random access (access to arbitrary concepts), though they were not considered in his work.

5 Conclusions

We raised the challenge of efficient learning and recognition of numerous concepts, and motivated the idea of a recall system to address these challenges. The ideal recall system, on each instance, quickly reduces the possible set of candidate categories of the instance to a small manageable set without missing the true concepts. We realized such a system by an index mapping features to concepts, specified an operationality criterion for the mapping, and gave an efficient online algorithm for learning it. We showed how online learning algorithms can utilize the recall system in order to efficiently learn classifiers for every concept. We demonstrated the promise of this framework on three large text categorization data sets. Future directions include developing a better understanding of the properties of

[7] In a multiclass problem, an instance belongs to exactly one category. So a multiclass solution method should be altered to handle the case when an instance can belong to multiple categories (*i.e.*, the multilabeled setting).

index learning algorithms, such as convergence and generalization properties, and insights onto why good operational indices may exist for natural learning problems. We plan to extend the techniques and to explore applications to other large-scale learning domains.

A general message of this paper is that efficiently learning an operational (fast and accurate) system is possible. The system was not programmed. We hope to see more work that follows this philosophy in tasks involving large-scale ongoing learning.

Acknowledgements

We thank Thomas Pierce who showed us how to use the Nutch search engine, and Yann LeCun for valuable pointers.

References

[AW98] S. Albers and J. Westbrook. Self-organizing data structures. In A. Fiat and G. Woeginger, editors, *Online Algorithms: The State of the Art*, pages 31–51. Springer LNCS 1442, 1998.

[CA98] Online Computation and Competitive Analysis. *A. Borodin and R. El-Yaniv*. Cambridge University Press, 1998.

[CBGTZ04] N. Cesa-Bianchi, C. Gentile, A. Tironi, and L. Zaniboni. Incremental algorithms for hierarchical classification. In *NIPS*, 2004.

[DB95] T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.

[DC00] S. Dumais and H. Chen. Hierarchical classification of web content. In *SIGIR*, 2000.

[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.

[FGL99] C. Fellbaum, J. Grabowski, and S. Landes. *WordNet: an Electronic Lexical Database*, chapter 9, Performance and Confidence in a Semantic Annotation Task. The MIT Press, 1999.

[GG06] S. Gangaputra and D. Geman. A design principle for coarse to fine classification. In *Proc. IEEE CVPR*, 2006.

[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.

[GSK05] K. Grill-Spector and N. Kanwisher. Visual recognition, as soon as you know it is there, you know what it is. *Pscychological Science*, 16(2):152–160, 2005.

[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-Verlag, 2001.

[IM98] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, 1998.

[Kel02] P. J. Kellman. *Handbook of Experimental Psychology*, chapter 7: Perceptual Learning. NY, Wiley, 2002.

[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.

[Mil95] G. Miller. WordNet: A lexical database. *Communications of ACM*, 1995.

[Mur02] G. L. Murphy. *The Big Book of Concepts*. MIT Press, 2002.

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

[RMG+76] E. Rosch, C. B. Mervis, W. Gray, D. Johnson, and P. Boyes-Braem. Basic objects in natural catagories. *Cognitive Psychology*, 8:382–439, 1976.

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

[RPH05] D. Ravichandran, P. Pantel, and E. Hovy. Randomized algorithms and NLP: Using locality sensitive hash functions for high speed noun clustering. In *Proceedings of Association for Computational Linguistics*, pages 622–629, 2005.

[RSW02] T. Rose, M. Stevenson, and M. Whitehead. The Reuters Corpus Volume 1 - from yesterday's news to tomorrow's language resources. In *Third International Conference on Language Resources and Evaluation*, 2002.

[RYA95] D. Roth, M-H Yang, and N. Ahuja. Learning to recognize 3D objects. *Neural Computation*, 14(5):1071–1104, 1995.

[Seb02] F. Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 2002.

[TFM22] S. Thorpe, D. Fize, and C. Marlot. Speed of processing in the human visual system. *Nature*, 381, 520-522.

[Val94] L. Valiant. *Circuits of the Mind*. New York: Oxford University Press, 1994.

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

[WW99] J. Weston and C. Watkins. Support vector machines for multi-class pattern recognition. In *ESANN*, pages 219–224, 1999.