Scaling Up Text Classification for Large File Systems

George Forman, Shyamsundar Rajaram
HP Laboratories
HPL-2008-29R1

Keyword(s):
machine learning, classification, document categorization, information retrieval, enterprise scalability, forensic search

Abstract:
We combine the speed and scalability of information retrieval with the generally superior classification accuracy offered by machine learning, yielding a two-phase text classifier that can scale to very large document corpora. We investigate the effect of different methods of formulating the query from the training set, as well as varying the query size. In empirical tests on the Reuters RCV1 corpus of 806,000 documents, we find runtime was easily reduced by a factor of 27x, with a somewhat surprising *gain* in F-measure compared with traditional text classification.

External Posting Date: June 21, 2008 [Fulltext] Approved for External Publication

Internal Posting Date: June 21, 2008 [Fulltext]

To be presented and published in the 14th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD'08), August 2008

Scaling Up Text Classification for Large File Systems

George Forman
Hewlett-Packard Labs
Palo Alto, CA, USA
ghforman@hpl.hp.com

Shyamsundar Rajaram
Hewlett-Packard Labs
Palo Alto, CA, USA
shyam.rajaram@hp.com

ABSTRACT

We combine the speed and scalability of information retrieval with the generally superior classification accuracy offered by machine learning, yielding a two-phase text classifier that can scale to very large document corpora. We investigate the effect of different methods of formulating the query from the training set, as well as varying the query size. In empirical tests on the Reuters RCV1 corpus of 806,000 documents, we find runtime was easily reduced by a factor of 27x, with a somewhat surprising *gain* in F-measure compared with traditional text classification.

Categories and Subject Descriptors

I.5.2 [**Pattern Recognition**]: Design Methodology— *classifier design and evaluation, feature evaluation and selection*; H.3.3 **[Information Storage and Retrieval]**: Information Search and Retrieval—*query formulation, selection process.*

General Terms

Algorithms, Performance, Experimentation.

Keywords

machine learning, text classification, document categorization, information retrieval, enterprise scalability, forensic search.

1. INTRODUCTION

Consider using a trained document classifier to search for 'relevant' files from one's personal file system—typically containing hundreds of thousands of files—or from within an enterprise containing billions of files spread across distributed servers worldwide. For such scale, it would be nearly infeasible to pump every single file through the document classifier. Yet such scalability could become essential for future Information Lifecycle Management (ILM) applications seeking to verify, for example, that corporate retention and file protection policies are being followed for certain classes of confidential documents. Likewise, this scalability could be demanded by future e-discovery or forensic searches to find all files related to a legal matter.

Note that in such applications, the objective is both precision *and recall*. This is in contrast to most information retrieval settings where only high precision in the top few search results is needed.

Information retrieval methods excel in scalability, as evidenced by their success in web search engines. However, text classification via machine learning is generally called for if one needs to balance precision and recall, assuming a training set is available. Note that for large scale corpora, the time to train the classifier is dwarfed by the cumulative classification time. The computational workload of classification is linear in the number of documents to be classified: each document is fetched from disk, its text features extracted, and then the classifier makes its class prediction. Even were thousands of CPU cores cheaply available to classify documents in parallel, it would place tremendous bandwidth demands on the disks and the I/O paths.

In this paper we improve the scalability of text classification by leveraging a full-text index over the corpus of documents. (The availability of such indices is becoming more common in personal and corporate file systems.) The basic concept is simple: we first use the index to quickly extract a small subset of documents that are potentially relevant, and then pass only these to the traditional text classifier. The workload of such a classifier is proportional to the size of the query hit list, yielding excellent speedup in the common case where only a small fraction of the documents are sought. This enables the system to scale up to very large document corpora. Our overall purpose is to optimize the design choices appropriate for querying one or more file systems, each with its own static full-text index.

Our research objective is to minimize runtime while maximizing F-measure—the harmonic average of precision and recall. The research questions include how to generate an effective query from the training set, how large a query is ideal, and how great is the savings in time vs. the tradeoff in accuracy? Although we expected a tradeoff, it turns out that *the two-phase process can be both much faster and more accurate* than a single text classification pass over all the documents.

For reproducibility, our experiments use publically available data and software. We use 140 classes of the large Reuters RCV1 corpus [9] indexed by Apache Lucene software v2.2.0 [6], and classified via the Weka v3.4 linear Support Vector Machine (SVM) model [13]. We take care that the operating system begins each timing experiment with a cold file cache, so that we accurately measure the performance of the whole system. A typical machine learning experiment conducts many runs in succession, but for this sort of information retrieval experiment, if one does not clear the cache, the pertinent data becomes cached in RAM, hiding the substantial cost of slow disk seeks.

Section 2 describes the problem scope, and Section 3 describes a variety of design choices around our solution. Sections 4 and 5 summarize a suite of experiments we performed. Section 7 discusses related work, and Section 8 concludes and offers perspective on future work.

2. THE PROBLEM SETTING

For the applications we are interested in, the volume of classification workload dwarfs the initial training time. This is especially so in view of recent breakthroughs in training state-of-the-art linear Support Vector Machine (SVM) models for text classification in near linear time, e.g. [7]. The rise of multi-core parallelism can aid with training time, but has little impact on the classification process, which is fundamentally I/O bound in fetching files for classification.

We expect that the full-text indices have been previously constructed for other purposes, with no special attention to the classification labels or tasks for which we leverage them.

Our research scope is limited to binary text classification tasks where the positive class of interest is usually rare, e.g. <1% of the population. Such high class imbalance is often a difficult operating region for machine learning. In particular, if the positive training examples are too rare, the learning process may select a decision threshold that classifies all items as negative, optimizing its training accuracy under uncertainty. Common techniques used in such situations are to over-sample the positives in training or to under-sample the negatives [11].

At some point after indexing, a training set is provided with labeled positive and negative training examples from which to learn a classifier to select all relevant documents from the entire collection. In practical settings, it is often not feasible to get a training set that represents a true random sample of the population of all files. For one thing, since positives are rare, asking a domain expert to provide binary labels to a stream of random samples is not an effective way to obtain a sufficiently large set of positive examples. If 1% of a random sample is positive, a domain expert would have to consider 10,000 randomly selected training cases to build up a set of 100 positive examples—many fewer would likely under-represent the diversity of positives. In practice, positive training cases are sometimes gathered by various *ad hoc* keyword searches, or have already been gathered in a directory by someone with an unknown, organic method. Considering this, the percentage of positives may be over-represented in the training set, but this is tantamount to under-sampling training negatives anyway [11].

3. TWO-PHASE CLASSIFICATION

Our approach consists of two phases: The first phase executes a query against a full-text index to determine a list of filenames that are likely positive. The second phase retrieves the file contents for each specified file, extracts its feature vector from the text content, and then classifies it. We list a number of design choices for each phase.

Phase 1: First, what is the space of query terms that may be used? Most text classification research focuses on the universal bag-of-words representation, although it has been shown repeatedly that including phrases can help substantially. In phase 1, we may only query for terms that have been previously indexed. Even so, most indexing packages provide the ability—at some additional overhead—to query for phrases. This gives us the flexibility to form our query from words only, or also to include phrases—adjacent pairs of words. We experimented with both options.

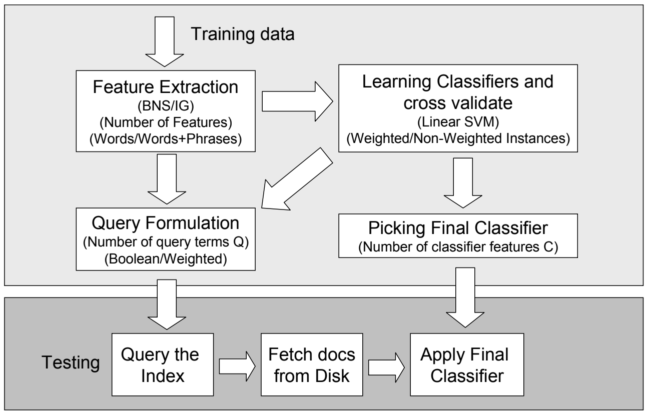


Figure 1. Block diagram of training and test phase where the design choices involved in each task are listed.

Next, given the large number of terms in the training set, how shall we select the best terms? And how many terms Q should we include in the query? We vary Q widely (1–16K), and evaluate term goodness via Bi-Normal Separation (BNS, our default) or via Information Gain (IG, only where stated)—two feature selection methods that have been shown to perform well [5]. We ignore terms that occur fewer than three times in the training set.

The computation for BNS is simply $|F^{-1}(tpr) - F^{-1}(fpr)|$, where F^{-1} represents the inverse cumulative Normal function from statistical tables, tpr represents the "true positive rate" of the feature, i.e. what percentage of positives the feature occurs in, and fpr represents the "false positive rate" of the feature—what percentage of negatives the feature occurs in. The computation for IG is better known and more involved:

$$IG = H(pos,neg) - [P(term)\, H(tp,fp) + (1-P(term))\, H(fn,tn)]$$

where

pos = number of positive training cases (minority),
neg = number of negative training cases,
tp = number of positive training cases containing term,
fp = number of negative training cases containing term,
fn = pos – tp,
tn = neg – fp,
P(term) = the prevalence of the term (tp+fp)/(pos+neg),
entropy $H(x,y) = -nln(x/(x+y)) - nln(y/(x+y))$, and
$nln(x) = x \log_2 x$.

Next, we have a design choice for the form of the query. We experiment with the two logical choices: (1) a straightforward Boolean query, namely Lucene's default disjunction[1] of the chosen terms, or (2) a weighted query, where each term is associated with a real number. The latter amounts to a linear classifier, especially if we allow the weights to be negative and only select documents that end up having a positive score. This opens the issue of how to set the weights. To limit our scope, we assume—not unreasonably—that a linear SVM provides state-of-the-art classification accuracy [7]. Thus, to obtain the weights for Q terms (whether words only, or words and phrases), we filter the training set so that it contains only the Q best terms according to BNS or IG, train a linear SVM classifier Weka v3.4 SMO using

[1] Although Google, Microsoft and Yahoo! Search each default to a *conjunction* of terms, such a query focuses on precision to the exclusion of recall and would be unworkable for our phase 1.

default parameters, and then extract its learned weight for each term. This will naturally include negative weights, e.g. for features whose presence is correlated with the negative class.

Our last option in phase 1 is whether to just focus on providing the best F-measure, or to try to bias toward higher recall and expect the phase 2 classifier to restore the precision to optimize our objective, F-measure.

Phase 2: We have the same choices with respect to the feature set to provide for training the final classifier. That is, we can optionally include phrases, select via BNS or IG, and can control the number of classifier features C used to train the final classifier. Except where explicitly stated otherwise, our typical phase 2 classifier used C=16,384 features selected via BNS from among the set of words and two-word phrases; these choices were selected after some preliminary experimentation. Because in phase 2 we have the complete file contents in memory, we can cheaply afford to use many features as far as it improves F-measure. More generally, phase 2 is not restricted to indexed features, so it could easily include other feature generators for improved accuracy, such as n-grams or domain-specific features. We leave this option for future work.

A final consideration is what training set to use for the phase 2 classifier. It will only encounter the files that were classified positive by phase 1, correctly or incorrectly. It seems proper then to train the phase 2 classifier only on labeled training cases that the phase 1 classifier finds positive. But it turns out this is impractical. There is a limited supply of training data, and the phase 1 classifier mostly excludes the negative training examples. Hence, the phase 2 training set would have most of the original positives, but would only sometimes contain a handful of negatives. It would be impractical in our setting to perform phase 1, and then ask the user to label a new large set of negatives from the query hits. Hence, we are left with the simple option to train phase 2 on the full training set.

4. EXPERIMENT METHODOLOGY

For a publically available corpus that includes ground truth classification labels, we use Reuters RCV1 [9], which has 806,791 news articles in XML formatted text files. We removed from all these files the metadata tags that reveal their true class labels, and saved this information in an isolated file. The average file size is 4 KB, which closely matches typical file systems historically [3]. We indexed all the Reuters files using the default Lucene text analyzer, which does not give any special consideration to the XML structure of the text. The indexing took just over two hours, including index optimization. Because indexing is slow, we did not consider rebuilding the index for each run to exclude the files used in the training set of that run. Hence, the training positives are among the query hits found in each run. (In a real deployment, it may well be the case that the training examples are already included in the index of the local PC; but this is less likely for a federated search of many distributed file systems throughout an enterprise.) Despite training examples being included among the query hits, we explicitly remove them before phase 2 and whenever we compute F-measure performance, in accordance with accepted practice for measuring performance in machine learning research. Recall that F-measure is the harmonic average of precision and recall: 2*p*r/(p+r). It drops rapidly if either precision or recall is poor.

Except where stated otherwise, each data point we show represents results averaged over a large set of separate classification tasks, i.e. macro-averaged. We selected all Reuters classes that have a prevalence <= 5% positive overall and have over 1000 examples (500 for training and over 500 others to find). This leads to 140 classes in all, ranging from 1001 to 37,410 examples, 6854 on average (0.1% to 4.6% positive, averaging 0.8% positive). The classes include Reuters geography/country codes, industry codes, and topic codes.

In each training set, we provide 500 positive examples and 5000 negative examples, selected at random just once from the ground truth labels. We want to be sure to provide enough training data to learn a decent classifier, so that we might avoid potentially useless '*garbage-in, garbage-out'* results. That said, for many classes, decent discrimination could have been learned with fewer examples. Exploring these tradeoffs is outside the scope of this paper. The query-time benefits of our methods are largely independent of the size of the training set.

Note that each training set has 9% positives, whereas the actual prevalence is typically ~1%. (We briefly tried training with 49,500 negatives to match 1% positive, but the Weka software crashed when it exhausted the 2 GB of available heap memory.) Instead of having a huge set of negative examples, we set the Weka instance weights such that the total weight of the positives amounts to 1% (alternately, some SVM implementations let one adjust the relative misclassification costs of positives vs. negatives). We do this for all phase 1 classifiers we train. Assuming the phase 1 classifier achieves decent precision, the phase 2 classifier should expect a much higher rate of positives. Thus, there is no need for weighting the training data for the phase 2 classifier. We confirmed this experimentally.

As mentioned in the introduction, it is important that we clear the file cache between tests, otherwise realistic disk delays are completely hidden. The ability for a user with root privileges to drop the file cache has recently been added to the Linux 2.6.16 kernel via "`echo 3 >/proc/sys/vm/drop_caches`". (Even so, this novel capability is still buggy as of 2.6.18-8.el5 and causes CPU soft lockups occasionally, requiring power-cycle reboots.) Specifically, we drop the cache before each query. We verified that without dropping the cache, we get wildly erroneous timings.

Hardware: HP Proliant DL360 G3 server, with dual 2.8GHz Xeon CPUs and 4GB RAM. It has a locally attached disk: a 36.4 GB, 10K RPM Ultra320 SCSI disk with an HP SmartArray 5i controller. We actually used 20 such servers independently to complete the many experiments involved; there was no communication or interference between them.

5. EMPIRICAL RESULTS

Our first set of results present the main take-home message of this paper: that two-phase classification greatly improves the speed as well as the final accuracy, compared to the baseline of simply testing every file with the (phase 2) classifier. Refer to Figure 2, which shows the overall F-measure on the y-axis, and the total elapsed time on the x-axis. The elapsed time includes the time to run the query, fetch the file contents for each query hit (excluding training cases), and extract its text features. Each file is effectively classified with no additional time at the completion of its text feature extraction.

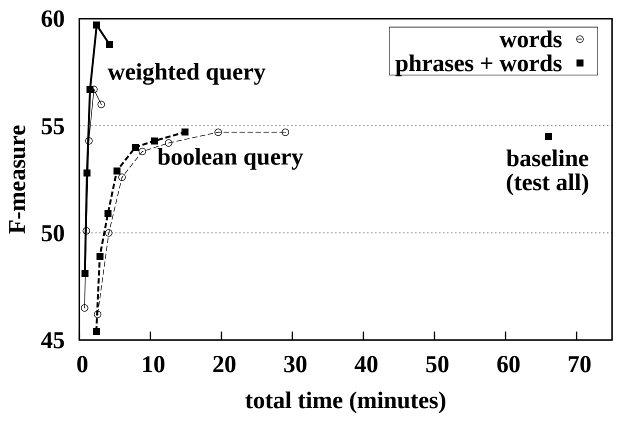


Figure 2. F-measure vs. elapsed time for various methods.

On the far right, we see the baseline method took ~66 minutes on average, and achieved 0.545 F-measure averaged over all 140 classes. The baseline method consists of the phase 2 classifier (16,384 words and phrases selected via BNS) applied to every file except the 5500 training files. Since no phase 1 is involved, the baseline classifier includes reweighting the training positives, which brings up its precision and F-measure substantially.

To the left in the graph, we see that Boolean queries of words alone (or words and phrases together) can greatly cut down on the number of documents to process in phase 2. The different points climbing up each of these curves correspond to Q=1, 2, 4, 8, 16, 32, or 64 query terms. With enough query terms, 100% recall is achieved on the positive class, and the phase 2 classifier obtains the same baseline F-measure, but 2–3 times faster. As Q increases, we see a rapid increase in the elapsed time: Once we have achieved 100% recall, additional query terms only serve to increase the number of false positives that need to be discarded by the phase 2 classifier. Lastly we note that because phrases are more specific, recalling fewer documents each, we see that more terms are required to achieve a given level of recall compared to the words only curve.

The pair of curves furthest left indicates the greatly improved overall performance of using a weighted query for phase 1. The different points represent Q=16, 64, 256, 1024, or 4096 terms, as selected by BNS, and the weights are determined from a trained SVM. Despite the extremely large number of query terms to process, in most cases we see much improved speed vs. the Boolean query—the weights give the phase 1 classifier much better control to exclude negatives while selecting positives. This increased precision cuts down on the irrelevant files that must be retrieved for phase 2. Furthermore, this more accurate, weighted phase 1 classifier excludes some negatives that otherwise get past the phase 2 classifier. Because it eliminates some complementary negatives, the effect is that the two-phase classifier obtains higher precision overall, improving the final F-measure average for all 140 classes. By using Q=1024 word & phrase terms in the weighted query, the process completes in just 2.4 minutes on average—27x faster than the baseline—with an F-measure 0.597 averaged over all 140 classes.

Comparing these two curves, we see that including phrases consistently improves performance at any given number of query terms. This effect is known, although most text classification research is done with a simple bag-of-words only.

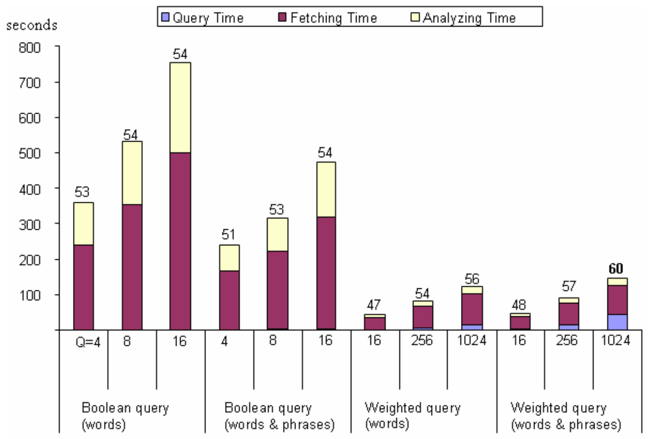


Figure 3. Time taken by different methods for varying query sizes. The time columns have been segmented into querying, fetching and analyzing time. Overall F-measure for each setting is shown atop each bar.

We made sure to extend Q far enough to verify that having more terms is not always better. This is consistent with feature selection literature, which usually shows a benefit to limiting the number of terms, e.g. [5]. But note that the weighted classifier can benefit from many more terms than the Boolean classifier (Q=1024 vs. ~32). Having such a large number of query terms slows down the query, but yields a speedup overall because phase 1 is more discriminating in which files to fetch for phase 2. As a result, large Q values lead to great savings in overall retrieval time. The best performing weighted query setting is 3x faster than the best performing Boolean query setting while achieving 10% better F-measure (6 points).

5.1 Timing Breakdown

Next, we break down the elapsed time of the x-axis of Figure 2 into its constituent parts: the time taken (1) to query the index, (2) to fetch the files that satisfy the query, and (3) to analyze the file contents for specific text features and thereby obtain its final classification. These times correspond to the three segments in each column of Figure 3. Boolean queries achieve nearly 100% recall with few query terms, and hence the query time is too small to see with respect to the total time. Boolean queries produce a large number of false positives, which leads to very high fetching and analyzing time. The fetching time averaged 12 ms per file, and the analysis time averaged 3 ms per file. Given that we end up fetching and analyzing thousands of files, it is relatively cheap to increase Q: an additional 7 ms per word on average, or 29 ms if we allow phrases. This relatively low incremental cost of adding terms opens an opportunity for weighted queries. They can be much more accurate, but they require significantly more query terms for good performance. The right half of Figure 3 shows weighted queries up to Q=1024 terms, where we begin to see the query time take a visually perceptible amount of the overall time. And because of their superior accuracy to Boolean queries, they waste much less time fetching and analyzing false positives. Figure 4 further illustrates this effect: For weighted queries there is a sharp increase in phase 1 recall rate with very large Q, and yet with very little increase in false positives. By contrast, Boolean queries obtain high recall at a terrible cost in false positives, i.e. bad precision.

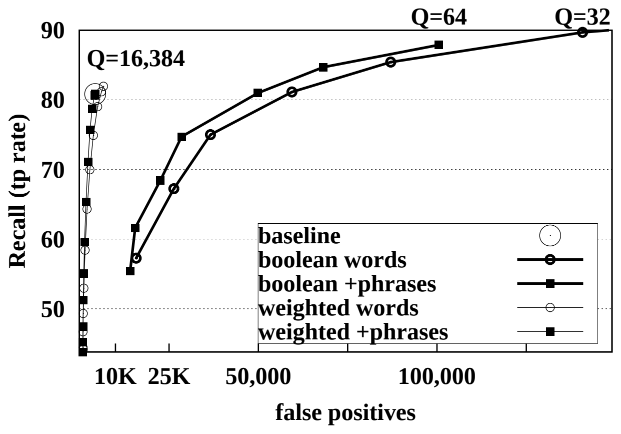


Figure 4. Recall vs. false positives for phase 1.

A natural question that arises at this point is whether the complete classification task can be done more efficiently in a single phase 1 pass. That is, given the trained linear SVM text classifier to be applied to the entire corpus, extract its weights and execute it only on the search engine, with no follow-up phase to further test the files. This basic idea has been tried [1], but not compared to the baseline, nor to two-phase classification. Note that the phase 2 classifier does not pay a time penalty for having a large number of features: the file fetch time and the feature extraction time depend on the disk performance and the file size, *not on the number of terms to be extracted for classification*. In our experiments, we found that 16K word and phrase terms was superior for the phase 2 classifier. But executing a query with such a large number of terms would pay a significant time penalty. In fact, we conducted this experiment and found it took 450 seconds, averaged over the 140 classes. This is 3x slower than using our two-phase system with Q=1024 in the first phase and 16K terms in the second. With such a large Q, the query time greatly exceeds the time it would take to fetch the few likely positive files and classify them. The two-phase classifier performs a balancing act in terms of the querying time vs. fetching and analyzing time.

But besides time, there is a further disadvantage to running a single, high-dimensional classification on the search engine: recall the baseline classifier did not achieve as good F-measure as the two-phase system. We discuss this effect next.

5.2 Cascaded Classifiers

Cascaded classifiers have been used extensively in face detection from images where there is a huge computational cost involved in determining for every window in an image whether it contains a face or not [12][10]. The computational load is overcome by cascading several classifiers, where the complexity of classifiers increases as we go further down the cascade. The first few classifiers of the cascade, which are very cheap, help in removing most of the "easy" negatives, and the more accurate, complex classifiers at the end of the cascade polish up with excellent discrimination, yielding good overall performance. Our approach in this work is similar in spirit, where we additionally use the index to quicken the early classification phase. Figure 5 illustrates the impact of our two-phase classification scheme in terms of F-measure, precision and recall. The x-axis varies the number of word+phase terms used, while the number of features in the final classifier is 16K, which yielded excellent performance on average. Two-phase classification has the consistent effect of

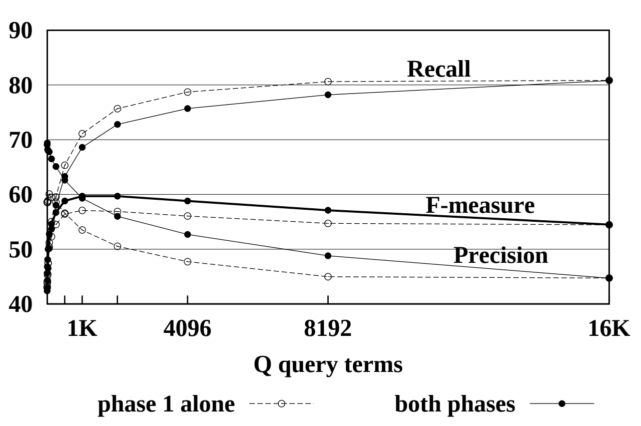


Figure 5. Effect of two-phase classification on each performance measure.

improving precision and lowering recall. This is natural, since a case will be classified negative if either classifier rejects it. The improvement in precision is generally more than the decrease in recall, which is reflected in the overall increase in F-measure. This can be attributed to the low correlation of classification errors of the two classifiers in the cascade which has been well studied in [8]. The average F-measure is highest at Q=1024 query terms. Note that at the far right point, where Q matches the number of terms in the final classifier, that the two-phase classifier has no effect. In this case, the two phases are computing the same function in different ways, and the final decisions match that of a traditional, one phase classifier. The benefit of the two phases only happens when the two classifiers have a somewhat different perspective on the training data. We have run additional experiments (not shown, but we could if the reviewers request) that vary the number of features in phase 1 and phase 2 independently, and they find a consistent plummet in F-measure whenever the number of features matches.

Figure 6 illustrates the two-phase classifier effect in terms of average F-measure over different groups of Reuters categories. The plots indicate a consistent F-measure improvement across all categories by using the two-phase classifier. Figure 6 also illustrates the F-measure variation based on the query term count Q. For the country based categories, good F-measure performance is obtained with just 64 query terms. On the other hand, the industry and the economy categories require a lot more query terms, around 1024 to achieve good F-measure. The impact of the two-phase classifier is also more pronounced in these difficult cases.

5.3 Policies for Choosing Parameters

All the experimental results shown so far present the F-measure averaged over the 140 classes (or some subset) for different parameter settings (Q and C). In a real-world setting, we are interested in picking parameters that maximize the F-measure for the single class at hand. We adopt a dynamic scheme using the cross-validation performance measures obtained during the learning phase. The cross-validation results of performance measures such as F-measure, precision and recall can be used to devise policies for identifying good parameter settings for each phase separately. We limit Q to be a power of 2 <= 1024 terms, in order to avoid inordinate query time. Table 1 shows the F-measure obtained for some of the top performing policies. The

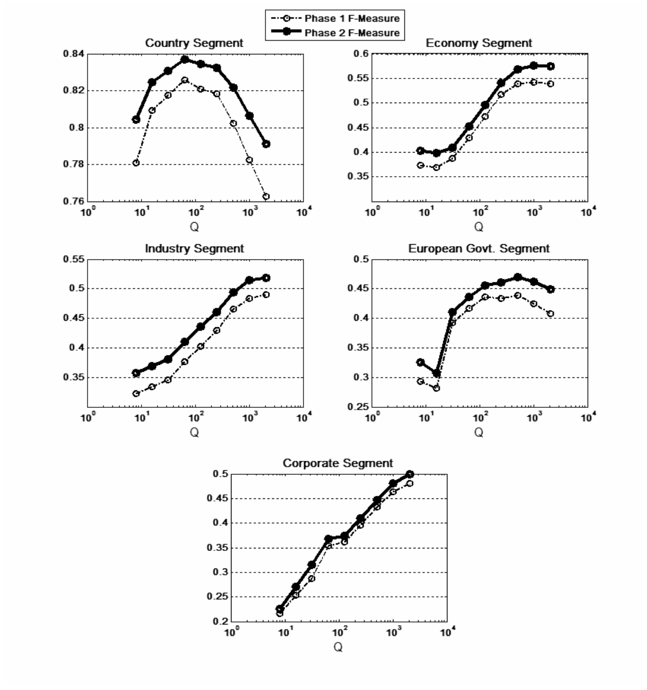


Figure 6. Average F-measure for different Reuters categories

results indicate that the best performing cross-validation based method is very similar to an oracle having access to the F-measure test set averaged over all 140 classes. Oracle I has access to the F-measures on test data for each class and hence, picks a parameter setting that maximizes the F-measure for each class. Clearly, this represents the highest achievable F-measure in this setting. Oracle II has access to the F-measure averaged over all classes for all possible settings.

5.4 Varying Design Choices

In Section 3 we described a palette of design choices for the phase 1 and phase 2 classifiers. Here we briefly present their effect.

Table 1: Comparison of F-measure obtained through cross-validation based policies to choose parameters compared with two different oracle methods.

Policy for Q	Policy for C	F-measure
Oracle I	Oracle I	62.6
Oracle II	Oracle II	60.0
Maximize F-measure	Maximize Precision	60.1
Maximize Recall	Maximize Precision	59.5
Maximize F-measure	Maximize F-measure	60.0

Weighting Instances to Approximate the Test Distribution:

A practical problem that arises in machine learning is one of changing class distributions from training to test phase, although it is typically avoided in most machine learning research [4]. As mentioned earlier, this problem arises in our problem setting as well. The class distribution of the test data of the first phase classifier is different from its training set. We adopt the approach of weighting instances appropriately to overcome the difference. While training the first phase classifiers, we weight the instances in such a way that the total weight of the positive training set is 1%—a rough estimate of the prevalence we expect for typical searches. (It would give an unfair advantage to determine the exact prevalence of positives in the test set and then set the weight exactly. Yet, this might be estimated via a quantifier [4].) The positive effect of such a weighting on the first phase F-measure is shown in Figure 7. On deeper inspection, the reweighting is consistently improving the precision of the classifier, naturally.

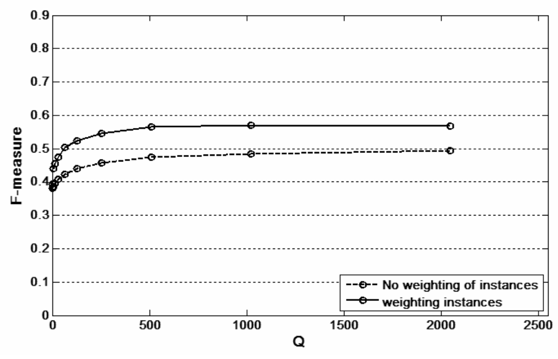


Figure 7. Weighting instances improves phase 1 F-measure.

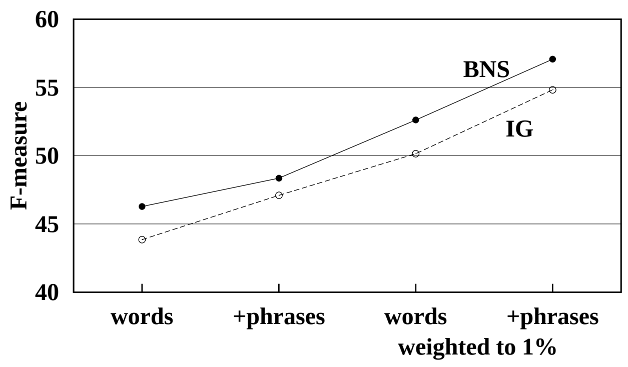
Phase 1 Performance:

Figure 8 shows the F-measure of the first phase alone, as we vary the cross-product of the design choices. For clarity of presentation , we hold Q=1024 fixed—a reasonably good choice for all. In this visualization, it is clear that (a) BNS consistently outperformed IG, consistent with past studies [5], (b) weighting the training positives to 1% significantly outperformed using the more balanced training set provided (right vs. left), and (c) the addition of phrases to the available terms improves performance. A paired t-test indicates very strong statistical significance, even between the two closest points (BNS vs. IG with phrases and without 1% weighting).

In addition to these differences, we also analyzed the effect on query time (not depicted). We found that IG queries consistently took longer than BNS queries, e.g. 2x slower. This is because IG tends to select terms that are more common and therefore have longer posting lists to process: 2.9x longer on average. The preference of BNS for rarer features is known [5], but here we have exposed a side benefit: it selects terms with shorter posting lists that are speedier information retrieval queries (as the research of [2] sought to do with IG-hybrid methods that somewhat preferred terms with shorter posting lists).

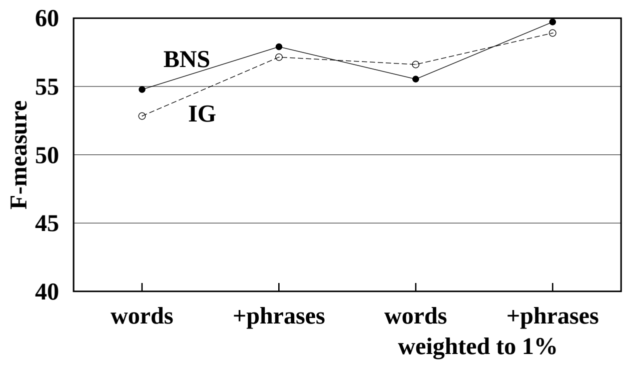
Phase 2 Performance:

These results so far only indicate the F-measure of phase 1. One might reasonably wonder whether these differences are mirrored by the final F-measure after both phases. Figure 9 presents the final F-measure as we vary the cross-product of the design choices (holding fixed Q=1024 query terms and C=16K classifier terms). Each design choice is kept consistent between phase 1



Phase 1 choices

Figure 8. Phase 1 F-measure vs. design choices.



Phase 1 choice = Phase 2 choice

Figure 9. Final F-measure vs. design choices.

and phase 2. (One could conceive of the cross product of phase 1 and phase 2 design choices separately, but such high dimensional results are expensive to compute and difficult to present.) The leading design choice (BNS +phrases and 1% weighted) continues to dominate, but the picture has changed somewhat. All the F-measures have improved vs. the F-measures of phase 1 in Figure 8. The superiority of each design choice for phase 1 is now less pronounced, but each is still present.

One design choice we discussed early on was to focus the phase 1 query on high recall, and expect the phase 2 classifier to increase the precision to the point that we obtain optimal F-measure. We experimented with such a strategy by leaving off the 1% weighting of the positive training examples; the 9% prevalence of positives in the training set therefore make the phase 1 classifier less conservative, and it yields higher recall. But then we found the phase 2 classifier was not able to bring the precision as high. We found that the highest average F-measure obtained was 57.9%, which is inferior compared to the F-measure obtained in the weighted case.

6. DISCUSSION

We have shown in this work that usage of phrases improves the F-measure of the classifier over a simple bag of words representation. Clearly, there are other features—such as n-grams, file metadata, file-format-sensitive features (e.g. stripping XML before indexing), and domain-specific features—which could potentially aid in improving classification performance and can be added to the index. On the other hand, there are cases where the file needs to be explicitly fetched. Consider a case, where someone is searching for articles about XML on their disk, or info about a merger with XML Inc. company. The index, being made with a generic bag of words parser that does not first strip XML, would have XML tags in every single Reuters article. Clearly, a query working on the index would do badly in terms of precision and this scenario furthers the case for having a second phase classifier which operates on file-format specific features, e.g. parse the XML and make features from the non-tag text.

Typical search engines today limit the number of query terms allowed to 20 or 30 maximum. Viewed from their perspective, they want to limit the resource consumption of customer queries. But as businesses begin to get greater value from their (central or someday, federated) search infrastructure via text classification queries, there will be business justification to raise term limits into the hundreds or thousands. This leads to a qualitatively different operating region for search engines, and we may see a substantial increase in computer resources used for text classification searches.

7. RELATED WORK

There is a single prior work in the literature that is highly relevant. Anagnostopoulos, Broder, and Punera [1] considered a classifier executed via a weighted search engine query, which they refer to as Weak AND (WAND). This is equivalent to our weighted queries of phase 1 alone. Their concern was with using a search engine for classification, with no phase 2 follow-up to improve accuracy. Our experiments in section 5.1 found that operating entirely on the index took 3x longer than if we included a separate phase 2. Even with respect to phase 1 alone, there are a number of dimensions in which our work adds further contributions: They only consider a bag-of-words model, where we also measure the benefit of phrases, which search engines easily facilitate. They only consider up to Q=100 terms, where we extend to Q=16,384. They did not consider BNS, which we found superior to IG. Their results are reported in terms of area under the ROC curve, which is mainly insensitive to improvements in F-measure when positives are very rare, e.g. 1%. And significantly, they did not run their experiments with a cold cache, so their reported results do not take disk performance into account. Additionally, we address the class distribution difference between the training set and test set using weighting of training data instances and we show significant improvement in F-measure performance. They consider the novel angle of biasing the feature selection according to the length of the term posting lists, in order to reduce processing time.

There is an older paper by Broder *et al* [2] that provides techniques for speeding up a traditional information retrieval query. Their experiments showed substantial speedup for queries averaging 2.46 to 7 terms, suffering little loss in precision. Coincidentally, they also use two phases, but the similarity is superficial. Their first phase retrieves the posting lists of the more important terms (as determined by their method), and the second phase retrieves all the remaining terms, but tracking scores only for documents that scored well in the first phase. Note that the work does not address classification, but their technique could be leveraged to speed up our phase 1 query on the index with Q≈1024. Our current software simply scores every document that is mentioned by the posting lists.

8. CONCLUSION AND FUTURE WORK

Our goal was to make it text classification scalable enough to scan distributed enterprise file systems for relevant documents. Ideally such an application would also include infrastructure to efficiently federate the search over many full-text indices throughout a corporation. In a real-world deployment, the size of the datasets would be much larger than the publicly available Reuters RCV1 collection, which occupies only 3.5 GB of disk space, and whose Lucene index occupies only ~500 MB of disk space. The index is small enough to fit entirely in RAM, although by dropping the file cache, we ensured that we were measuring realistic disk delays. For real-world usage, the index would be much larger and the reverse posting lists would be longer. While this suggests longer query times and perhaps a desire to reduce the query size Q to save time, keep in mind that with a larger, distributed infrastructure also comes much longer latency and lower bandwidth to fetch the actual files for phase 2. Hence, if anything, we anticipate the time and workload savings of our two-phase technique to be proportionately more important for larger environments. The World Wide Web is an extreme thereof—a query to a fast, central index server such as Google can efficiently provide a list of relevant documents, but actually fetching them for a second phase test will be very slow. Moreover, on a large scale it is likely to suffer from web servers being unavailable and intermittently poor network performance. In view of this, text classification on the web may best be served by extensive computation with the reverse indices and perhaps Google's large file cache. In the end, it may still be important to fetch and check the final list of hits, since URLs often become stale when web pages are deleted. Also, a web page may have changed substantially since its indexing, and no longer match the matching criterion.

An interesting direction of future research would be to devise schemes for using an index in an active learning setting. In the real-world where training data is so scarce, active learning plays a crucial part in acquiring labeled data through an oracle who has time constraints. In such a time constrained setting, it is crucial that the processing to identify useful training examples for labeling has to be done quickly. The index can be exploited to quicken the processing step, which normally involves identifying the most informative instance(s) to be labeled towards improving the current classifier.

In this paper, we excluded the training time from interest, for we are targeting settings where it is dwarfed by the volume to classify. But with active learning, this training time goes into the user's interactive loop. The rising multi-core revolution can decimate this training time, as well as the time to analyze file content for text features. However, it does not address the primary bottleneck for this application: disk seek time involved in executing queries and fetching files. In fact, the relative speed between CPU and disk is becoming larger, so the techniques described in this paper should continue to be relevant.

Considering the enormous interest in ranking problems, a related research direction would be to analyze the impact of two-phase processing through an index in the context of ranking. In our experiments, we noticed a consistent improvement in precision in the top 20 from first phase to second phase. It would be interesting to analyze the performance of two-phase ranking schemes on rigorous ranking accuracy measures.

9. ACKNOWLEDGMENTS

Our thanks to Hernan Laffitte and Eric Anderson for their invaluable support in setting up racks of machines with the updated Linux kernel.

10. REFERENCES

[1] Anagnostopoulos, A., Broder, A. Z., and Punera, K. 2006. Effective and efficient classification on a search-engine model. In *Proc. of the 15th ACM International Conference on Information and Knowledge Management* (Arlington, VA, Nov. 6-11, 2006). CIKM '06. ACM, 208-217.

[2] Broder, A. Z., Carmel, D., Herscovici, M., Soffer, A., and Zien, J. 2003. Efficient query evaluation using a two-level retrieval process. In *Proc. of the Twelfth Int'l Conference on information and Knowledge Management* (New Orleans, LA, Nov. 03 - 08, 2003). CIKM '03. ACM, 426-434.

[3] Douceur, J. R. and Bolosky, W. J. 1999. A large-scale study of file-system contents. *SIGMETRICS Perform. Eval. Rev.* 27, 1 (Jun. 1999), 59-70.

[4] Forman, G. 2006. Quantifying trends accurately despite classifier error and class imbalance. In *Proc. of the 12th ACM Int'l Conf. on Knowledge Discovery and Data Mining* (Philadelphia, Aug. 20-23, 2006). KDD'06. ACM, 157-166.

[5] Forman, G. 2003. An extensive empirical study of feature selection metrics for text classification. *J. Machine Learning Research.* 3 (Mar. 2003), 1289-1305.

[6] Hatcher, E. and Gospodnetic, O. 2004 *Lucene in Action (In Action Series)*. Manning Publications Co.

[7] Joachims, T. 2006. Training linear SVMs in linear time. In *Proceedings of the 12th ACM SIGKDD Int'l Conference on Knowledge Discovery and Data Mining* (Philadelphia, PA, Aug. 20-23, 2006). KDD '06. ACM, 217-226.

[8] Kittler, J., Hatef, M., Duin, R. P. W., and Matas, J. 1998. On combining classifiers. *IEEE Trans. On Pattern Analysis and Machine Intelligence*, vol.20, no.3, Mar. 1998.

[9] Lewis, D. D.; Yang, Y.; Rose, T.; and Li, F. 2004. RCV1: a new benchmark collection for text categorization research. *J. Machine Learning Research,* 5:361-397.

[10] Luo, H. 2005. Optimization design of cascaded classifiers. In *Proc. of the 2005 IEEE Computer Society Conf. on Computer Vision and Pattern Recognition (CVPR'05) – Vol. 1* (June 20-26, 2005). IEEE Computer Society, 480-485.

[11] Van Hulse, J., Khoshgoftaar, T. M., and Napolitano, A. 2007. Experimental perspectives on learning from imbalanced data. In *Proceedings of the 24th International Conference on Machine Learning* (Corvalis, Oregon, June 20 - 24, 2007). ICML '07, vol. 227. ACM, 935-942.

[12] Viola, P. and Jones, M. J. 2002. Robust real-time object detection. *International Journal of Computer Vision.*

[13] Witten, I. and Frank, E. 2005. *Data Mining: Practical machine learning tools and techniques*, 2nd Edition, Morgan Kaufmann, San Francisco.