

Plan for Fall 2002 Software Development for Experiments on Approximate Nearest Neighbor Text Classification

David D. Lewis
(21-Jan-2003 Draft)

I. Introduction

We have two proposals for using approximate nearest neighbor methods to reduce the disk space, peak RAM usage, and/or compute time requirements of nearest neighbor (NN) text classification systems, while preserving effectiveness.

This document is a high level discussion, including some design details, of the software that would need to be developed to test these approaches, comparing them to standard baseline techniques.

....

II. How NN Text Classification Works

There are two major NN approaches to text classification:

1. Traditional
2. Weighted

Both methods use a set of **training** documents, and a set of topics. Each training document has a **label set** for each topic. The label set contains one **label** for each topic, with a value of either "Yes" (document is relevant for topic) or "No" (document is nonrelevant for topic). "No" labels will be stored implicitly for now (V.C.1). "Unknown" labels may be used in the future, possibly displacing "No" as the implicit value (VIII.D).

II.A. High Level View of NN Text Classification Methods

II.A.1. Traditional kNN

Traditional kNN classifiers classify a test document as follows:

(1) Find the **k-neighbors** for test document *v* (i.e. the *k* training documents with the highest similarity to a test document or, if there are fewer than *k* training documents, all training documents).

- (2) For each topic t ,
 - (a) Find the number, $p(v,t)$, of the k -neighbors have a "Yes" label for that topic.
 - (b) If $p(v,t)$ is greater than a threshold value $b(t)$, assign topic t to the document.

We will set k manually for now (see V.D.2.), though in the future we may support adapting it (VIII.K.). We will allow $b(t)$ to be set either manually (a single value for all topics), or to be tuned on a per topic basis (V.D.).

II.A.2. Weighted

Weighted kNN classifiers classify a test document as follows:

- (1) Find the k -neighbors for test document v .
- (2) For each topic t ,
 - (a) Compute a score $s(v,t)$ of test document v for topic t , taking into account the similarities of neighbors which do or don't have a "Yes" label for topic t .
 - (b) If $s(v,t)$ is greater than a threshold value $b(t)$, assign topic t to the document.

The thresholds $b(t)$ are often tuned on the training documents, and often vary across topics.

There are many variant ways to compute the score in (2a). Yiming Yang has done the most work on these approaches, so as a start I am proposing we focus on replicating her results, though we will undoubtedly want to develop improved algorithms ourselves.

II.B. Details of Weighted NN Classification Approaches

Here are the main approaches Yang has used:

- i. Yang94 (Yang has variously called this "ExpNet" and "kNN")

$$s(v,t) = \text{SUM sim}(v,x) * l(x,t)$$

where $l(x,t)$ is 1.0 if test document x has label "Yes" for topic t , and 0.0 otherwise. Summation is over the k -neighbors of v .

- ii. Yang99 [Earlier reference is Yang, etal 1999, IEEE Int. Sys]

$$\begin{aligned} s(v,t) &= \text{SUM sim}(v,x)*l(x,t) - \text{SUM sim}(v,x)*(1-l(x,t)) \\ &= \text{SUM sim}(v,x)*(2*l(x,t) - 1) \end{aligned}$$

where $l(x,t)$ is as above and summation is again over k -neighbors.

- iii. Yang00a (Yang in SIGIR 2000 called this kNN.avg1)

$$s(v,t) = \frac{\text{SUM sim}(v,x)*l(x,t)}{\text{SUM } l(x,t)} - \frac{\text{SUM sim}(v,x)*(1 - l(x,t))}{\text{SUM } (1 - l(x,t))}$$

$$= \text{SUM sim}(v,x) * \left[\frac{l(x,t)}{\text{SUM } l(x,t)} - \frac{(1 - l(x,t))}{\text{SUM } (1 - l(x,t))} \right]$$

where $l(x,t)$ is as above and summation is again over neighbors. If similarity is dot product, this is equivalent to computing a local Rocchio model over the neighbors and using that model to score the example.

iv. Yang00b

$$s(v,t) = \frac{\text{SUM_P sim}(v,x)*l(x,t)}{\text{SUM_P } l(x,t)} - \frac{\text{SUM_N sim}(v,x)*(1 - l(x,t))}{\text{SUM_N } (1 - l(x,t))}$$

where $l(x,t)$ is as above, SUM_P is over the nearest k_P examples with $l(x,t) = 1$, and SUM_N is over the nearest k_N examples with $l(x,t) = 0$.

v. New02a

An obvious generalization of Yang00a is to introduce a weighting parameter as in the usual Rocchio algorithm:

$$s(v,t) = \text{beta} * \frac{\text{SUM sim}(v,x)*l(x,t)}{\text{SUM } l(x,t)} - \frac{\text{SUM sim}(v,x)*(1 - l(x,t))}{\text{SUM } (1 - l(x,t))}$$

vi. New02b

Similarly, we can generalize Yang00b

$$s(v,t) = \text{beta} * \frac{\text{SUM_P sim}(v,x)*l(x,t)}{\text{SUM_P } l(x,t)} - \frac{\text{SUM_N sim}(v,x)*(1 - l(x,t))}{\text{SUM_N } (1 - l(x,t))}$$

Of these Yang94, Yang00a, and New02a (of which Yang00a is a special case) are the ones I propose testing.

II.C. Advantages and Disadvantages of NN Text Classifiers

The primary advantages of NN text classifiers are:

1. Ability to fit any classification function given enough training data
2. Good (or less bad, anyway) efficiency when there are large numbers of topics

The second is because (for most NN classifier algorithms) the lookup of neighbors only needs to be done once, no matter how many topics there are. If labels are stored sparsely, compute time can be made independent of the number of topics.

Disadvantages of NN text classifiers are:

1. Less effective than best approaches
2. Inefficient with small number of topics
3. Sensitive to quality of term weighting

II.D. NN Text Classifiers with Small Amounts of Training Data

Most research on NN text classification, like most research on text classification, has used relatively large amounts of training data. The main exception is Yang's work on the TREC adaptive filtering track and the TDT tracking task. [DDL: I will be looking at this work more closely to see if there are additional algorithms of interest to us.]

II.E. Online Learning of NN Text Classifiers

There has been little research on online learning of NN classifiers. Indeed, implementing the Traditional NN classifier with a manually chosen neighborhood size and threshold in an online fashion is trivial: just tack the new example on the end of the list of examples.

For us, however, online learning poses three challenges:

1. Most NN_Classifier variants we use require tuning a threshold on the labeled data. This is expensive for NN classifiers in any case (V.D.), and worse if it must be done for each new labeled example.
2. Some NN_Lookup methods index the training data in a way that is not easy to incrementally update. It may be necessary to define hybrid methods that cache examples in a Direct_NN_Lookup, and only periodically reindex.
3. We may want to compute corpus statistics (e.g. IDF) using new data (both labeled and unlabeled). For some term weighting methods this affects only the test vector, but for others it affects the training vectors as well.

Our current plan is to delay work on online learning of NN classifiers until January.

III. Computing Similarities

There are two main issues: text representation and the

similarity function.

III.A. Text Representation

For simplicity, and to reduce the need for tuning, I suggest we use the same text representation as in most of Yang's experiments: remove words on the SMART stopword list, Porter stemming, log TF x IDF weighting, cosine normalization. We'll omit feature selection for simplicity, as it doesn't improve effectiveness much, as long as IDF is used.

III.B. Similarity Function

We will use dot product as our similarity function. We sometimes, but not always, will normalize both test and training vectors to have a 2-norm of 1.0, in which case dot product corresponds to cosine correlation.

The NN approximation schemes proposed by Rafael and Martin approximate Euclidean distance rather than dot product. Fortunately, we can compute dot product from Euclidean distance:

$$u \cdot v = 0.5 (||u||^2 + ||v||^2 - ||u-v||^2)$$

I haven't worked out how additive and relative approximation errors on the Euclidean distance impact dot product.

IV. A Proposed Object-Oriented Approach to NN Classification

The section proposes a preliminary object-oriented (OO) design for NN text classifiers. Two primary kinds of objects are used:

1. NN_Lookup
2. NN_Classifier

Each NN_Classifier uses an object belonging to a subclass of NN_Lookup. The NN_Lookup subclasses have methods to find neighbors of a vector, and do computations on neighbors, in an exact or approximate way.

In particular, all NN_Lookup subclasses must implement the method Find_Neighbor_Labels_and_DotProducts(), which can support any NN_Classifier type. Some NN_Lookups provide a richer interface that allows the NN_Classifier to do less work.

V. Further Design Issues

V.A. Implementation of NN_Lookup Classes

V.A.0. NN_Lookup (parent)

The Parent NN_Lookup has pure virtual methods for `Index()` and `Find_Neighbor_Labels_and_DotProducts()`. Each subclass is required to implement these two methods as a minimum. It may implement other methods as well.

I propose that the parent NN_Lookup provide explicit implementations of `Threshold_Label_Counts`, `Threshold_Label_DotProduct_Sums`, `Threshold_Label_DotProduct_Difference_of_Means` based on calling `Find_Neighbor_Labels_and_DotProducts()` and doing the obvious thing. [DDL to Vladimir: Note this is a change from my opinion during our discussion last week.]

Optionally, NN_Lookup could implement the other methods discussed in VI.A. as well, probably in this fashion:

3a. `Find_Neighbor_Labels_and_DotProducts` : NOT IMPLEMENTED HERE
3b. `Find_Neighbor_Labels` : CALLS 3a
3c. `Find_Label_Counts`: CALLS 3b
3d. `Threshold_Label_Counts`: CALLS 3c
3e. `Find_Label_DotProduct_Sums` : CALLS 3a
3f. `Threshold_Label_DotProduct_Sums` : CALLS 3e
3g. `Find_Label_DotProduct_Difference_of_Means`: CALLS 3a
3h. `Threshold_Label_DotProduct_Difference_of_Means`: CALLS 3g

Since all functionality here bottoms out at `Find_Neighbor_Labels_and_DotProducts`, these versions would not be as efficient as special purpose ones. Subclasses can always override these versions. However, I believe by using the above calling sequence they would pick up the most closely suited efficient implementation supported by that child.

While we aren't implementing `Insert()` yet, when we do I suspect it should be a pure virtual for the Parent NN_Lookup class.

V.A.1. Direct_NN_Lookup

This implements `Index()` and `Find_Neighbor_Labels_and_DotProducts()`.

`Index()` stores the training vectors and their associated label sets in an unordered list or array. `Insert()` is optional, but would simply add a training vector to the end of that list.

`Find_Neighbor_Labels_and_DotProducts()` explicitly computes the similarity between the test vector and each

training vector, sorts the similarities, and returns (a pointer/ref to) a list of label set/similarity pairs.

V.A.2. Inverted_NN_Lookup

This implements `Index()` and `Find_Neighbor_Labels_and_DotProducts()`.

`Index()` builds an inverted index, i.e. a listing for each term (vector coordinate) of which training examples have a nonzero value on that coordinate, and what that value is. It separately records the label sets associated with each example.

`Find_Neighbor_Labels_and_DotProducts` then can compute the similarity of each training example using only those inverted lists for which the test example has a nonzero term value. An accumulator is used to total the score of each training example. (A hash table could in theory be used to avoid having an accumulator for training examples with a similarity of 0.0 anyway, but this is probably not worth the effort.)

V.A.3. Heuristic_Inverted_NN_Lookup

[DDL: I'm still looking into which of a large number of possible approximate nearest neighbor calculations using inverted files would be appropriate for us.]

V.A.4. Sketched_NN_Lookup

`Index()` creates a sketch (see Martin's description) for each training example, as well as storing the label set and the 2-norm (see below) for each example.

`Find_Neighbor_Labels_and_DotProducts` computes the approximate Euclidean distance from the test example to each training example using the sketches. From that approximate Euclidean distance it then computes an approximate dot product, taking the 2-norms of the test and training example into account (III.B).

V.B. Implementation of NN_Classifier Classes

An `NN_Classifier` class has at least these responsibilities:

- Handing the test vector it's given to an `NN_Lookup` class and getting information on neighbors back.
- Returning a set of predicted class labels for the test vector to the caller.

`NN_Classifier` classes vary in what additional work they need to do.

V.B.1. NN_Classifier (parent)

It seems likely that per-topic thresholding should be supported at this level, since all plausible subclasses would find it useful.

V.B.2. Traditional_NN_Classifier (parent)

This class implements the traditional NN classifier. It can operate in one of two ways:

1. If a manual threshold on $p(v,t)$ is specified, that means all topics use that same threshold. That threshold should be passed to `Threshold_Label_Counts` as the default threshold.
2. If thresholds are to be tuned, then for each topic a separate threshold is found (see V.D.), and this set of thresholds is passed to `Threshold_Label_Counts`.

V.B.3. Yang94_NN_Classifier (parent)

This class implements the Yang94 approach. It always tunes a threshold for each topic. The actual classification is done by `Threshold_Label_DotProduct_Sums()`.

V.B.4. New02a_NN_Classifier (parent)

This class implements the Yang94 approach. It always tunes a threshold for each topic. The actual classification is done by `Threshold_Label_DotProduct_Difference_of_Means()`.

V.C. Implementation of Misc. Classes

V.C.1. Label_Set, thresholds, and Related Objects

`Label_Set`, threshold objects, and other objects storing information on topics should take advantage of sparseness of labels.

The neighbors of a test vector typically include "Yes" labels for only a few topics. NN methods should do computation proportional to the number of topics with "Yes" labels among the neighbors, not the total number of topics among all training examples.

For instance, we might use a hash table in the threshold object passed to `Threshold_Label_Counts()`, `Threshold_Label_DotProduct_Sums()`, and `Threshold_Label_DotProduct_Difference_of_Means()` (see VI.A.). One would look up thresholds only for topics that have nonzero scores, i.e. have at least one "Yes" among the neighbors. This works if we forbid thresholds ≤ 0 . We can use either $>$ or $\geq$ when comparing scores to thresholds; current implementation uses $\geq$.

V.D. Tuning Thresholds

All NN classification methods (except Traditional with a manually chosen threshold) require tuning a separate threshold $b(t)$ for each topic. The straightforward approach does this:

```
FOR each topic
  FOR each training example u
    Find its score  $s(u,t)$  for this topic
  Sort the score/label combinations
  Find optimal threshold for desired effectiveness measure
```

Unfortunately, computing $s(u,t)$ requires a call to some method of NN_Lookup (e.g. Find_Label_DotProduct_Difference_of_Means), and the above loop requires $T*N$ such calls, where T is the number of topics and N is the number of training examples. Each call to the NN_Lookup method takes, for Direct_NN_Lookup, $O(N*(L+M))$ computation, where L is the average number of nonzero terms in a vector, and M is the average number of topics assigned to a vector. We then have the cost of sorting for each topic, so $O(TN \log N)$. So total computation is $O(T*N*N*(L+M))$. For 1000 topics with 3 training examples each, a plausible count of operations, NOT including sorting, would be $1000*3000*3000*100$ or 900 trillion operations. Not plausible.

An obvious improvement is:

```
FOR each training example u
  Find  $s(u,t)$  for all topics  $t$  where  $s(u,t) > 0$ 
FOR each topic t
  Sort the score/label combinations
  Find optimal threshold for desired effectiveness measure
```

This requires only $O(N)$ calls to, say, Find_Label_DotProduct_Difference_of_Means, so total cost using Direct_NN_Lookup, is $O(N*N*(L+M) + T*N \log N)$. This is better, since we lose a factor of T in the first term, at the cost of having the nonzero $s(u,t)$'s for all topics in memory at the same time.

But we still need improvements to make this practical.

There are two places to look for them:

1. Replacing Direct_NN_Lookup with something faster, which is in any case the goal of the current work.
2. Making sorting and thresholding faster by some of the techniques we discussed in context of Rocchio.

See VIII.L. for other future thresholding issues.

V.E. Configuration File Options

The configuration file will specify which version of NN_Classifier and NN_Lookup are used in a particular run, as well as any parameters they take.

V.D.1. Configuration Options for NN_Lookup Classes

NN_Lookup : Can't be used directly.

Direct>NN_Lookup: no additional parameters

Inverted>NN_Lookup: no additional parameters

Heuristic>NN_Lookup: [DDL: To be determined.]

Sketched>NN_Lookup: [DDL to Vladimir: I will start an email conversation among you, me, and Martin to determine what the parameters controlling the quality of the approximation should be.]

V.D.2. Configuration Options for NN_Classifier Classes

i. Traditional>NN_Classifier:

a. "Number_of_Neighbors" : A real value ≥ 0.0 .

b. "Fixed_Threshold" or "Learn_Threshold_For_Each_Topic": If "Fixed_Threshold" is specified, a threshold real value ≥ 0.0 must also be specified. (There are no additional arguments if "Learn_Threshold_For_Each_Topic" is specified.)

ii. Yang94>NN_Classifier:

a. "Number_of_Neighbors" : A real value ≥ 0.0 .

b. "Learn_Threshold_For_Each_Topic" (V.D.) is the only allowed thresholding option this classifier supports. (So we probably don't want to require the user to actually include this in the config file.)

iii. New02a>NN_Classifier:

a. "Number_of_Neighbors" : A real value ≥ 0.0 .

b. "Learn_Threshold_For_Each_Topic" is the only allowed thresholding option this classifier supports.

c. "NN_Rocchio_Beta" : A real value ≥ 0.0 . (Note: beta=1 corresponds to Yang00b.)

VI. Methods Provided by NN_Lookup and its Subclasses

This section describes the NN_Lookup methods in more detail. The only methods that we require a subclass of NN_Lookup to implement are Index() and Find_Neighbor_Labels_and_DotProducts(). All other methods

are currently allowed to raise an exception. See VI.B. for why we mention them at all.

In addition to the required parameters listed for each method, the implementation of a method for a particular NN_Lookup subclass may have additional parameters (e.g. controlling quality of approximation).

VI.A. Method Descriptions

1. Index:

INPUT: List of (training vector, label set) pairs

RESULT: Create data structures necessary to implement methods 2a and 3a on the specified pairs, plus any optional methods chosen. Any already existing data structures are first erased.

2a. [OPTIONAL] Insert

INPUT: (training vector, label set) pair

RESULTS: Update data structures with new vector, such that method calls using those data structures produce results that approximate those produced if Index() was called on all training vectors received so far.

3a. Find_Neighbor_Labels_and_DotProducts:

INPUT: Test vector v; integer k.

OUTPUT: Set of (Label_Set, similarity) pairs that approximates the corresponding set for the k-neighbors of v. "Approximates" is left deliberately vague. See 3b to 3h for some of the things the approximation might be used for.

3b. [OPTIONAL] Find_Neighbor_Labels:

INPUT: Test vector v; integer k.

OUTPUT: Set of Label_Sets that approximates the corresponding set for the k-neighbors of v.

3c. [OPTIONAL] Find_Label_Counts:

INPUT: Test vector v; integer k.

OUTPUT: For each topic t , a pair $(t, e(t))$.

$e(t)$ should be an estimate of how many of the k -neighbors of v have label $l(x,t) = 1$. Estimates of 0 should be represented implicitly, i.e. if $e(t) = 0$, omit the pair.

3d. [OPTIONAL] Threshold_Label_Counts:

INPUT: Test vector v ; integer k ; thresholds B .

B specifies a default integer-valued threshold d , and a list of pairs of the form $(t, b(t))$ where t is a topic id and $b(t)$ is an integer threshold value. Any topic not explicitly represented in B is assumed to have default threshold d .

OUTPUT: For each topic t , a pair $(t, f(t))$.

$f(t)$ should be True if the system believes, with confidence specified by approximation parameters, that at least $b(t)$ of the k -neighbors have $l(x,t) = 1$. Omit pairs with $f(t) = \text{False}$.

3e. [OPTIONAL] Find_Label_DotProduct_Sums

INPUT: Unlabeled test vector v ; integer k .

OUTPUT: For each topic t , a pair $(t, e(t))$.

$e(t)$ should approximate the sum of $\text{sim}(v,x)$ over those k -neighbors x that have $l(x,t) = 1$. Omit pairs with $e(t) = 0$.

3f. [OPTIONAL] Threshold_Label_DotProduct_Sums

INPUT: Test vector v ; integer k ; thresholds B .

Thresholds are as in 3d, but real-valued instead of integral.

OUTPUT: For each topic t , a pair $(t, f(t))$.

$f(t)$ should be True if system estimates, with confidence specified by approximation parameters, that $e(t) \geq b(t)$, where $e(t)$ is also defined in 3e. Omit pairs with $p(t) = \text{False}$.

3g. [OPTIONAL] Find_Label_DotProduct_Difference_of_Means:

INPUT: Test vector v ; integer k ; real β .

OUTPUT: For each topic t , a pair $(t, e(t))$

$e(t)$ should approximate β times the mean of $\text{sim}(v, x)$ for those k -neighbors which have $l(x, t) = 1$, *minus* the mean of $\text{sim}(v, x)$ for those k -neighbors with $l(x, t) = 0$. (See description of Yang00a and New00a for more details). Omit pairs with $e(t) = 0$.

3h. [OPTIONAL] Threshold_Label_DotProduct_Difference_of_Means:

INPUT: Test vector v ; integer k ; thresholds B ; real β .

OUTPUT: For each topic t , a pair $(t, f(t))$.

$f(t)$ should be True if the NN_Lookup object estimates, with confidence specified by approximation parameters, that $e(t) \geq b(t)$. $e(t)$ is as defined in 3g. Omit pairs with $f(t) = \text{False}$.

VI.B. The Optional Methods

We don't require implementing Insert() for now, but it will be a high priority early next year, since the focus of our project is on online filtering.

Methods 3b to 3h provide an interface that is more restricted than that of 3a. The rationale for including them is:

- 1) The restricted interface is sufficient for, and indeed is motivated by, the needs of one or more of the NN classification methods, and
- 2) It may be possible to make better approximations to these restricted use of neighbors than to the general uses supported by 3a.

In fact, my recommendation is for the top level NN_Lookup class to implement at least these three optional methods, which correspond to the natural interfaces for the three main classifier variants we will test:

Traditional_NN_Classifier : 3d (Threshold_Label_Counts)
 Yang94 : 3f (Threshold_Label_DotProduct_Sums)
 Yang00a, New02a : 3h (Threshold_Label_DotProduct_Difference_of_Means)

The computation done by these methods would otherwise need to be done by the appropriate NN_Classifier object, so we might as well include it in NN_Lookup to allow reuse. (However, I'm open to arguments for the alternative approach.)

VII. Methods Provided by NN_Classifier and its Subclasses

Here's a tentative specification of methods for NN_Classifier. I haven't thought these out as carefully as those for NN_Lookup.

VII.A. Methods

1. Initialize()

INPUT: Parameters for NN_Classifier

RESULT: Resets state, remembers parameters.

2. Train()

INPUT: List of (training vector, label set) pairs

RESULT: The vectors and labels are indexed in an NN_Lookup. Thresholds are then tuned against the labeled data if the NN_Classifier uses dynamic thresholds.

3. Classify()

INPUT: Unlabeled test vector v

OUTPUT: For each topic t, a pair (t, f(t)).

f(t) should be True if NN_Classifier decides the vector belongs to this class. Pairs with f(t) = False should be omitted, so all f(t)'s will be True.

VII.B. Future

Additional methods will be necessary when we start to support online learning.

VIII. Issues For Future Work

In addition to the issues here, see Section II.E. on online learning.

VIII.A. Serialize/Unserialize Methods

As described above, each NN experiment must start by loading the training data, building any indexing structures, and then running on the test data. For methods where the Index() method is expensive, it would be desirable to be able to save and restore the index data structures. Even when Index() doesn't do anything but save the data, there would still be an advantage to loading a small set of examples from a small file, rather than having to look them up in a large file, and an advantage to not having to load both the examples and the judgment files.

VIII.B. Delete / Relabel / Reweight Methods

The API presented above supports adding new examples to the NN

classifier. However, it would also be desirable to support deleting examples, changing the labels of examples, and/or changing the label weights of examples (see VIII.E).

VIII.C. Term Weighting Variants, Including Weighting Dimensions

It would be desirable try alternatives to cosine normalization for document length, to make sure that the approximate NN algorithms are not implicitly relying on vectors having a 2-norm of 1.0.

Some of these document length variants (e.g. pivoted normalization) allow corpus weighting (e.g. IDF) and feature selection to be handled separately from document vector construction and normalization. This makes the system more flexible and will be very desirable to meet our goals for incorporating prior knowledge. It will also raise interesting new approximation issues.

VIII.D. Finding Neighbors within a Specified Subset of Documents

There are several situations where we need not the k nearest documents, but the k nearest documents that also satisfy some other property:

Case 1. "Unknown" Labels: Our initial experiment will assume each training document has a "Yes" or "No" label for each topic. More realistically, there would be a small number of documents, consisting of relevant documents and near misses, which are labeled for each topic. Training documents would have "Unknown" labels for most other topics. Straightforward approaches to this case require we find, for each topic, the k nearest documents labeled for that topic. The union of these sets could be very large, possibly the whole set of documents. However, before we study efficiency issues in this case, we need to first study how well NN methods work at all in this situation. If NN is a reasonable approach here, efficiency issues are likely to be tricky. Using pseudolabeling (Section VIII.D.) may help.

Case 2. "Yang00b" : The Yang00b method requires the k_P closest relevant documents and the k_N closest nonrelevant documents for each topic. In this case, not even pseudolabeling avoids the need to potentially get a very large set of neighbors. An efficient approach here may require taking the threshold for each topic into account, in order to prune topics where the k_P closest could not in any case bring the score above threshold. Also, Yang00b is a relatively recent and ad hoc local learning procedure, and there may be alternatives that are both more effective and more efficient.

VIII.E. Weighted Labels

There's several situations (variable quality data, pseudolabeling, etc.) in which we want to associate weights with example/label pairs. (Weights would be associated with each label, rather than with an example, because a training document could be a good quality example for one topic and a poor quality one for another.)

Label weights could easily be used by both the Traditional and

Weighted NN algorithms, though I know of no research on this. We in particular will need to test whether neighborhood size should be based on number of neighbors or sum of weights of neighbors. Once we understand effectiveness of weighted labels better, we can look at efficiency issues. Introducing weighted labels would require a change to label sets (Section V.C.1.).

VIII.F. Pseudolabeling Data for Nearest Neighbor Classification

Pseudolabeling (using current classifier to artificially assign low weight simulated labels to unlabeled training documents) is sometimes useful, particularly for avoiding extreme classifier behaviors (reject all, accept all). Little is known about using pseudolabeling with NN classifiers, so again we'll need to do experiments.

Assuming the approach is useful, there's interesting approximation issues in pseudolabeling, including the possibility that it would be done implicitly during classification.

VIII.G. Taking Advantage of How Threshold Values Change or Don't Change

Some approaches keep thresholds constant over time and/or over topics. It may be possible to take this into account in the Index() method. This is a relatively low priority, as it is likely that for the better methods thresholds will vary over topics and over time.

VIII.H. Binary Vectors as Native Form of Document Vectors

Past work in IR shows that term weighting improves effectiveness very strongly, so restricting representations to binary vectors is not a good idea. However, our project is researching new binary text representations which may change this. Similarity measures like Hamming distance might then be useful.

VIII.I. Getting Vectors and/or IDs for Neighbors

The above design does not require that NN_Lookup be able to return the actual vectors for neighbors, only that it be able to do various computations on those neighbors. This allows approximation methods that do not keep the original vectors.

It may be desirable, however, to support this in some cases. For instance, suppose we had a way to efficiently return a set of 2k training vectors that are guaranteed (or highly likely) to contain the k neighbors. One could wrap this class in another that gets the 2k vectors, computes their exact similarities, and then does computations based on the exact set of k-neighbors.

It may also be desirable to allow a meaningful ID to optionally be associated with each training vector at index time, and to be able to return the IDs of neighbors or approximate neighbors.

VIII.J. NN Indexing Techniques Applied to Linear Classifiers

If we do develop interesting efficiency techniques in these

experiments, we might apply them to speeding up linear classification as well. Instead of finding the documents with highest dot product with a test vector, we could find the linear classifiers with highest dot product. The main difference is that each linear classifier has its own threshold - it would be as if thresholds were associated with training examples in NN rather than with topics. But this can be handled by extending the test vector with a feature that is always 1.0, and including a constant term in the classifier, making the threshold for each classifier be 0. Or the classifiers might be logistic regression models, in which case we'd be looking for high posterior log odds values.

The one difference from NN, is that we'd want to find all classifiers with similarities over a particular value, rather than the k nearest, whatever their similarity. But this capability would probably be useful for NN as well.

VIII.K. Adapting Neighborhood Sizes

Neighborhood size implicitly controls the complexity of classifiers (smaller neighborhoods correspond to more complex classifiers). One could imagine tuning this either for a set of topics or on a per-topic basis. There hasn't been much research on this, and there are interesting computational challenges as well.

VIII.L. Improving Thresholding

VIII.1. Avoiding Scoring All Examples

A deeper understanding of nearest neighbor classification may let us avoid having to compute scores for all training examples against each other. (Yes, that's rather vague at the moment.)

VIII.2. Leave One Out Cross Validation in Thresholding

One problem with thresholding as described in Section V.D. is that a training example contributes to its own score for purposes of computing thresholds. Of course, this was also true in computing thresholds for Rocchio classifiers, but potentially is more of a problem for NN, particularly if the neighborhood size is small. We may want to use leave-one-out cross-validation in such computations. This would be easy for most NN_Lookup methods, but potentially hard for Rafi-style approximation methods.