

Categorization for MALACH

J. Scott Olsson olsson@math.umd.edu

Progress report

Supervisor: Doug Oard, oard@umd.edu

December 12, 2003

Abstract

Work done on an implementation of a k Nearest Neighbors (KNN) algorithm for text classification is motivated and the implementation is summarized. Some preliminary results are briefly discussed. A plan for future work is outlined.

1 Motivation

The MALACH Project (Multilingual Access to Large spoken ArCHives) is a joint collaboration between researchers at UMD, JHU, IBM, and the Survivors of the Shoah Visual History Foundation (VHF), whose purpose is to “dramatically improve access to large multilingual collections of recorded speech in oral history archives” [8].

Shortly after the release of *Schindler’s List* in 1994, the VHF began collecting and archiving video testimonies from Holocaust survivors and witnesses. To date, more than 116,000 hours of this film (including some 52,000 persons) has been collected and archived in over 180 terabytes of MPEG-1 video. This data set, the “world’s largest coherent archive of videotaped oral histories” [1], represents fertile ground for research in automatic speech recognition, machine translation, natural language processing and information retrieval.

In order to provide useful access to large spoken archives, many challenging subproblems must be addressed. Figure 1 contains a graph of some of the larger component problems and their relation. If we are to provide useful methods for information retrieval, we must first classify the segmented data in the archive, which requires us to have segmented the textual data, which of course requires us to have transcribed the original video data to text. That is to say, a video testimony must first be transcribed to textual data, broken into small text segments, and categorized, before robust information retrieval can occur.

Categorization may be conceived as a supervised learning problem, wherein a collection of vectors (representing a bag¹ of words in documents) termed the *training set* has for each vector an associated set of labels

¹ie, A multiset (a finite unordered set which respects multiplicity)

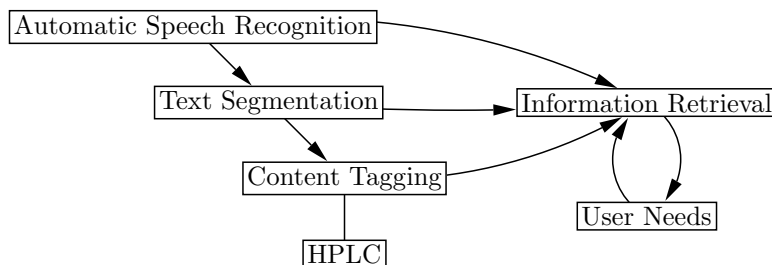


Figure 1: Several large subproblems of the MALACH project, and their relations.

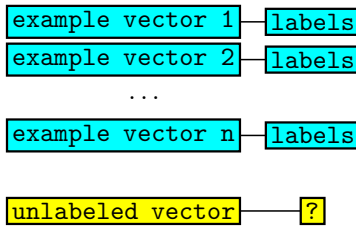


Figure 2: The vector space statement of the categorization problem, which naturally presents itself for non-hierarchical (flat) categorization problems. This is the view taken by kNN, while other models may state the problem differently (eg, documents may be viewed as mixtures of word probability distributions).

and the goal is to determine the labeling for other, unlabeled document vectors. These “unlabeled” vectors are termed the *test set*.²

The initial proposal addressed segment categorization in the MALACH project using a hierarchical probabilistic model as described by Guassier, et al[4]. Such a categorizer (a Hierarchical Probabilistic Latent Categorizer, or HPLC), will attempt to leverage the observable hierarchical structure in the MALACH dataset to improve categorization performance; this structure has not yet been accounted for, nor is it accounted for in the work here discussed. The work here discussed has the intent of instead providing a baseline for comparing the effectiveness of the kNN implementations developed at IBM and UMD, while also furthering our understanding of the dataset which will facilitate the development of the hierarchical model.

We begin by briefly restating the HPLC model, and then turn our attention to the focus of our work so far, a kNN implementation. The discussion of HPLC is here included for completeness and to ground the word done so far in view of future work.

2 The HPLC Method

2.1 The segment model

Documents are most often modeled using words as features and their frequency as the feature value. In HPLC, however, the modeled features are co-occurrences of words j in segments i , (i, j) .

The model assumes the data had been probabilistically generated in the following way:

1. Pick a segment class α with probability $P(\alpha)$,
2. Choose a segment i using the class-conditional probability $P(i|\alpha)$,
3. Sample a word topic ν with the class-conditional probability $P(\nu|\alpha)$.
4. Generate word j using the topic-conditional distribution $P(j|\nu)$.

A segment class α refers to a set of segments sharing some (here undefined) common thematic features (eg, the segments all refer to “the post war era” or “propaganda”), while a word topic ν denotes a semantic field which may be described by a particular word distribution. In this model, segments may be partly relevant to different topics, and segments from different classes may share words from topics in their common ancestry. From our knowledge of the data set, we expect some hierarchical structure to be exhibited (eg, Berlin descends from Germany which descends from World). Therefore, it is hoped such a (hierarchical) model will more accurately categorize the segments.

According to the model, the probability of a particular word/segment co-occurrence, $P(i, j)$, is given by:

$$P(i, j) = \sum_{\alpha} P(\alpha)P(i|\alpha) \sum_{\nu \uparrow \alpha} P(\nu|\alpha)P(j|\nu)$$

(where we have used the fact that segments and words are conditionally independent). The notation $\nu \uparrow \alpha$

²In fact, the test set is also labeled, although the labels are reserved for evaluation purposes.

signifies we sum over all ν which are ancestors (upwards) of α ; in effect, $P(\nu|\alpha) = 0$ when $\nu \not\uparrow\alpha$. The parameters of the model are then simply the discrete probabilities $P(\alpha)$, $P(i|\alpha)$, $P(\nu|\alpha)$, and $P(j|\nu)$.

2.2 Training

The goal of HPLC is to categorize a segment (expressed as a set of co-occurrences), given a training set of co-occurrences and their respective, manually assigned categories. During the training phase, the model parameters may be obtained by maximizing the likelihood, that is

$$P(i, j, \alpha) = P(\alpha)P(i|\alpha) \sum_{\nu} P(\nu|\alpha)P(j|\nu).$$

The maximum likelihood estimates for $P(\alpha)$ and $P(i|\alpha)$ may be trivially computed, while the remaining parameters cannot be obtained analytically. Instead, we will obtain them using an iterative Expectation-Maximization (EM) algorithm [2] with deterministic annealing [5, 7].

2.3 Categorization

A segment d may now be categorized based on the posterior class probability $P(\alpha|d) \propto P(d|\alpha)P(\alpha)$, where the class probability $P(\alpha)$ is known from the training phase and $P(d|\alpha)$ may be estimated using EM to maximize the log-likelihood of the segment w.r.t. $P(d|\alpha)$:

$$L(d) = \sum_s \log \left(\sum_{\alpha} P(\alpha)P(d|\alpha) \sum_{\nu} P(\nu|\alpha)P(j(s)|\nu) \right),$$

where s is the number of co-occurrences in the segment.

Once $P(d|\alpha)$ is estimated for every segment class α , we expect the segment to belong to the class for which $P(\alpha|d)$ is maximized.

3 The kNN method

3.1 The segment model

The k Nearest Neighbor model views documents as vectors, where the i^{th} vector element represents the salience of term i in the associated document. Of course, documents come as documents, not as vectors, and so steps must be taken to produce the vectors from the original document data. This process can be roughly broken into two phases: document mangling (which includes document acquisition and formatting as well as dimensionality reduction steps such as stoplisting and stemming) and term weighting. An identical procedure is followed for training and test document mangling, while similarly motivated but slightly different (in form) term weightings are chosen for each.

3.1.1 Document mangling

The MALACH scratchpad³ data is stored in a flat file of tab delimited entrees, one field of which contains the scratchpad entry (also present are the document's unique segment and interview identification numbers—used internally to, for example, associate category ID's with segments in memory). Punctuation is stripped and capitalization is removed. The bulk of this pre-processing is currently being done in Perl (as data formats often change and because, computationally, mangling represents a very small fraction of the categorization cost—all of which may be performed once before several independent categorization runs are made, having perhaps different runtime parameters).

³During the interviewing process, many MALACH interviewers took short notes (often summaries) for their own use. This metadata, referred to as the *scratchpad data*, is content rich and (it remains to be seen) will likely aid significantly in the categorization effort for the automatic speech recognition (ASR) data (of which we do not yet have sufficient data for testing).

3.1.2 Stoplisting

Stoplisting is the process of removing words which are expected to not contribute in any meaningful way to the semantic of a document (the canonical example being the word “the”). Stoplists (the list of words to be ignored) are most often (and here) produced by a statistical analysis of representative documents in the problem domain; in our case (as is commonly done) we simply stop on words which occur above some threshold frequency across the training collection. Some example stopwords used are: *the, about, he, she, and, from, and of*. These words are not expected to aid in categorization, and their removal can significantly reduce the size of the problem (both in terms of storage and computational cost).

3.1.3 Stemming

Stemming also helps reduce the dimensionality of the feature space considered. In stemming, portions of words (namely suffixes) which are not expected to contribute significantly to their meaning are stripped off (on the assumption that words like *happy* and *happiness* contribute in a similar way to a documents semantic). These are typically heuristic algorithms hand tweaked for a particular language (as morphological rules in English will differ significantly from those in Latin or Greek). This kNN implementation includes both a Porter and Lovins stemmer (the Lovins stemmer came “for free” with the YASE code base used for inverted file creation, while the Porter stemmer is more frequently used today and was therefore implemented in addition). To do this, I simply modified a freely available C implementation of the Porter stemmer to accept and return string data in the format used internally by the categorizer.⁴

Both stemmers can be chosen at compile time by a compiler flag (preprocessor directive) and are thus modular in the C sense.

3.2 Term weighting

In this step, the vectors representing documents in the training set, the *document term weight* vector (*DTW*), are constructed, where for each term i ,

$$DTW_i = (1 + \log(DTF_i))IDF_i$$

where DTF_i represents the document term frequency (the number of times term i appears within the particular document), and IDF_i , or the inverse document frequency, is a weighting which indicates how useful the i^{th} term is for distinguishing between documents in different classes. This is here calculated as

$$IDF_i = \log\left(1 + \frac{N}{TF_i}\right),$$

where N is the total number of documents in the collection and TF_i is the number of documents which term i appears at least one time in (the term frequency)—the intuition being that terms which appear in every document ($N/TF_i = 1$) will not be as useful for classifying as words which appear in only a small subset of the documents ($N/TF_i > 1$) and that a term which appears twice as often as another may be less useful than the latter, but not half as useful (ie, the weighting is logarithmically damped).

In kNN, term weights must also be calculated for documents in the test set (query documents) This follows a similar motivation as in the training vectors, although the weighting is slightly different. This weighting is currently implemented for each *QTW* (query term weight vector) as

$$QTW = \left(\frac{1}{2} + \left(\frac{1}{2} \frac{QTF}{QMF}\right)\right)IDF,$$

⁴ANSI C uses a pointer to type `char` and a trailing ‘\0’ delimiter to store text strings; I have chosen instead to implement them as a pointer to $N + 1$ unsigned bytes in sequential memory, where the first byte contains the number of bytes which follow (N). This has proven to simplify string manipulation (eg, superfluous calls to `strlen` can be avoided); I have also found it easier to keep track of memory allocation and garbage collection in this way.

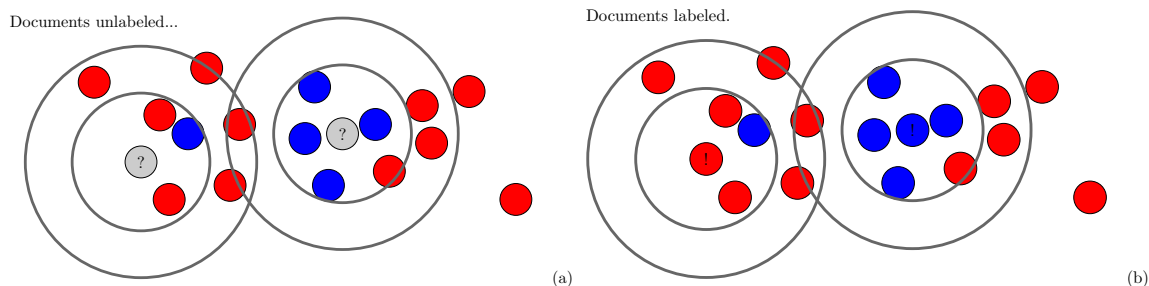


Figure 3: An imagined two dimensional projection of several labeled (red or blue) document vectors in feature space. In (a) the test documents are unlabeled, and in (b) they have taken the labels of their nearest neighbors in the space.

where IDF is used from the training phase⁵, QTF is the term frequency of a given term in the particular query document, and QMF is the maximum frequency of any term in the query.

Both the document and query term weighting schemes here described follow the method employed by the YASE project [6] (an open source search engine which I have used as a basis for my code, particularly to construct inverted document files). It is not clear *a priori* whether one term weighting scheme is superior to any other (to make such a decision from the fore, one would need a better understanding of the dataset's properties). The point here being that the effectiveness of one weighting scheme over another itself carries important information about the dataset's structure (perhaps, for example, an indication of the importance of the hierarchy). The teams at IBM and at UMD have chosen different weighting schemes (preliminary results indicate the performance of each is differing in significant ways), which will allow this knowledge base to grow more quickly (and hopefully, to maximize the effectiveness of a hierarchical classifier). The most significant difference between these kNN implementations is precisely this choice of term weighting.⁶

3.3 How kNN works

With the documents presented as vectors, we are now free to imagine each document as representing a data point in some n dimensional feature space, where n is the number of terms across the collection. The very simple idea of the nearest neighbor algorithm is to determine which document in the training set is nearest to the test document in the feature space, and to then assign the test document labels from that nearest neighbor. Figure 3 graphically depicts this idea. The algorithm generalizes as one might expect for $k > 1$ neighbors.

3.4 Similarity Measure

Vectors can be *near* each other in many senses; one could calculate their Euclidian distance, their overlap, or their inner product (perhaps with some normalization). I have chosen to do the latter, and calculate the similarity of a training and test document as

$$S = \frac{(DTW)^T(QTW)}{\|DTW\|_2},$$

so that, within a factor of $\frac{1}{\|QTW\|_2}$ (which has no effect, since the one query is constant across all the training documents for each categorization attempt), S is simply the cosine of the angle between DTW and QTW .

⁵Here as always, we assume that the training documents are representative of document in the domain of interest—here the test set.

⁶eg, The group at IBM is accounting for document length in the term weighting while I am accounting for it in the similarity measure (by cosine normalization).

The norm $\|DTW\|_2$ can be thought of as the length of the particular document vector (which enforces the intuitive constraint that a document with twice as many salient terms as another is not considered more similar than the other if it is simply much longer).

3.5 Class labeling

A similarity measure is calculated for a test document's query vector and every document in the training collection that shares at least one term in the query. As similarity scores are calculated, they are inserted into a balanced binary tree (AVL), which removes the need to later sort the scores (cf. Section 3.6.2 below). The k largest scores are then pulled from the tree and used to determine the test documents labels.

3.5.1 Training data acquisition and storage

The categorizer must be told which labels are mapped onto which training documents and then provide that data through an interface to other code portions (eg, the labeling code). This happens in the following way: 1) the training labels are available (after some possible preprocessing) in a delimited flat file which contains an entry for each segment ID and it's respective category numbers; this file is called the *ground truth* file. 2) the categorizer parses this file and builds a doubly linked list containing a key for each segment ID, as well as a pointer to a byte containing the number of category numbers (unsigned bytes) that follow in sequential memory.⁷ In the immediate future I will be switching this storage into a balanced binary tree (it had been implemented before a tree code), which has an element retrieval time of $O(\log N)$ (a significant improvement over the worst-case $O(N)$ of a list traversal). 3) This category data is then provided to the rest of the categorizer by a function call which includes a pointer to the doubly linked list as well as the segment idea (key) for which category data is requested; the function returns with a pointer to the first byte of the requested data (which, recall, stores the number of category numbers, or bytes, which follow in sequential memory).

At this point, we have k nearest training documents, their similarity scores, and their associated categories ID's.

Labeling now proceeds in the following way: 1) a new singly linked list of all the categories returned for the k nearest neighbors is created, wherein each list element contains the sum of all similarity scores associated with that element's respective category number. These sums of scores are termed the *category scores*. 2) All categories having a score above a *category score threshold* are taken as labels for the test document⁸.

3.6 Data structures

3.6.1 Lists

Linked lists⁹ have been implemented for storing category numbers and for tallying category scores. The category score tallying list is small (typically order 10 elements) and so is probably for our purposes a better choice than a faster tree structure having more creation overhead. On the other hand, storing category numbers for each segment ID in a list is clearly a poor choice¹⁰ (keep in mind there are 100,000 training

⁷This step could be done instead during the training phase, storing the category data in the inverted document file for each document ID (and in fact, this is how I originally implemented it); I have chosen rather to do it during categorization because the operation runs in constant time and we may conceivably change training labels between runs (and will then not want to reconstruct the inverted indexes of our training data).

⁸Contrast this to the simpler problem for which only one category is to be assigned per test document, in which case we might only take the category with the highest returned score.

⁹A linked list is simply a collection of allocated units in memory where each element contains a key, some data, and a link to the next element in the list. In principle, lists (in the abstract sense) could be stored as contiguous block of memory, but this would require the entire block to be reallocated if a list element was added beyond the size of the original block.

¹⁰It was implemented as a doubly linked list first simply because it was easy to do quickly; changing this over to a more suitable data structure will not be difficult, as these more suitable data structures have been put in place for other portions of the code.

segment ID's, ie. list elements, in our small initial problem, and that each traversal of the list is $O(N)$); a clearly better choice is a balanced binary trees.

3.6.2 AVL-Trees

An AVL (Adelson-Velskii and Landis) Tree[11] is a binary search tree¹¹ with a balancing condition¹²—that is, as elements are inserted into the tree, the tree is *rotated* to ensure that the node taken as root has (within 1) equal number of descendents on its right and left.

Such a tree is currently being used to store the similarity scores as they are computed; consequently, as we only desire the k largest scores and the tree is guaranteed to have a height of $O(\log_2 N)$, we can find our k nearest neighbors in only $O(k \log_2 N)$ time. Alternatively, the scores would have to be tallied in a list and then sorted, but the asymptotic best time for a sort is $O(N \log_2 N)$ —we can achieve this best possible runtime with less work by simply using a balanced tree.

3.6.3 Inverted document files

An inverted document file[10] is simply a sorted list of terms having links to the documents containing those terms as well as any associated term attributes (here, for example, the inverted document frequency). The YASE project [6] had already implemented an inverted document file, as a B-Tree, which I have retained. There is not presently motivation to replace or reconstruct this portion of the code, although my original approach required modifying the inverted document file structure (at that time, I was storing category numbers with segment ID's in the document database). The B-Tree structure is not optimal for small document collections (ie, collections for which the inverted document file can be stored in main memory), but will be preferable as the document collection grows (ie, as the inverted document file is increasingly resident in secondary storage). As the MALACH project moves to ASR data, these storage concerns will become increasingly important.

3.6.4 B-Trees

B-Trees are a common method of storing and constructing an inverted document file. A B-Tree of order M is defined as a tree with the following properties[11]: 1) the root is either a leaf or has between 2 and M children; 2) all nodes except the root have between $\lceil M/2 \rceil$ and M children. 3) all leaves are at the same depth (ie, all leaves have the same number of edges in the shortest traversal from the root to themselves). Because each node has more than 2 children, every traversal decision made is more complex than in a binary tree (wherein one can simply decide if the key you are looking for is greater or less than the key of the current node). Offsetting this decision cost is the fact that fewer total decisions will have to be made (ie, the depth of a leaf is shorter in a B-tree¹³ than in a binary tree). If each decision (logic) is cheaper than traversing an edge (a memory or, with large data structures, a disk access), a B-Tree will be preferable.

¹¹A binary tree is a simple, undirected, connected, acyclic graph with one node denoted the root; every node contains a key and has at most two edges departing from it (termed the *left* and *right* edges). Every descendent along a node's right edge will have only keys larger than the node's key, and likewise, every descendent along a node's left edge will have only keys smaller than the node's key. Elements which have no descendents are termed *leaves*. The *height* of the tree is the number of edges along the longest traversal from root to a leaf.

¹²One can imagine a worst case in which elements were inserted into a tree already sorted, so that the root would have the smallest key and successive elements always descended to the right; in other words, a list rather than tree would be built. A balanced tree then is simply a tree such that the root has an equal number of nodes descending on its right and left (within 1); a balanced tree ensures that the height of a full tree is $O(\log_2 (1 + N))$

¹³The depth of a B-Tree is at most $\lceil \log_{\lceil M/2 \rceil} N \rceil$.

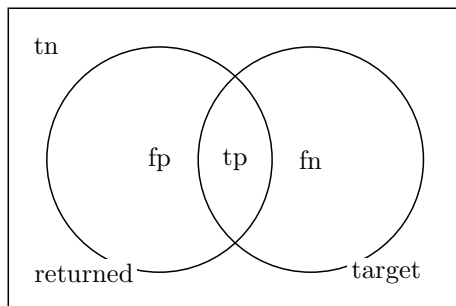


Figure 4: Possible results of classification attempts.

4 Validation/Evaluation

4.1 Definitions

Categorization attempts will fall into one of four groupings, given a target set (the set of all documents truly in a class) and a set of documents actually returned (ie, the set of documents classified—truly or falsely—as the target class). Accordingly, categorizers may fail in one of two different ways: 1. they may falsely identify a document as positive—believing it to be in the target class (fp), or 2. they may falsely identify a document as being negative—not in the target class (fn). Likewise, they may succeed by both determining a document is not in a class (tn) and by determining that a document actually is in the target set (tp).[3] Figure 4 shows the possible classification results more clearly.

Ultimately, a categorizer is only as good as the end user is happy with it’s output. That said, several measurements of quality are typically made for categorization attempts. The naive choice, the ratio of correct to incorrect responses (accuracy), turns out to be somewhat uninformative—as it makes no distinction between types of failures. Precision,

$$P = \frac{tp}{(tp + fp)},$$

the ratio of classified documents in the target class to all classified documents, also does not tell the entire story—as a categorizer may be made as precise as desired by only categorizing a small number of documents. Recall,

$$R = \frac{tp}{(tp + fn)},$$

the percentage of target documents properly classified, also falls short—simply classifying all documents in the collection as the target class would guarantee perfect recall!

Ideally, then, a system will have both high precision and high recall, although, in practice, one is typically gained at the expense of the other (they are typically roughly inversely proportional). In practice, then, some weighted average of the two measures is used to ascertain the effectiveness of the categorizer. Keep in mind that a given categorizer will generally only perform optimally if appropriate runtime parameters are set (eg, the number of neighbors, k , or the threshold score to consider in kNN), and so a single metric of overall suitability is desirable. The F measure is often used, and is defined as

$$F = \frac{1}{\alpha \frac{1}{P} + (1 - \alpha) \frac{1}{R}},$$

where α represents the relative weight value placed on precision and recall; an equal valuing ($\alpha = .5$) reduces the measure to the harmonic mean of P and R ,

$$F = \frac{2PR}{(R + P)}.$$

To calculate these measures, the categorizer builds a *contingency table* for each category decision¹⁴ made during the test phase. A contingency table is simply a tabulation of the cardinality of each partition shown in Figure 4, that is,

	yes is correct	no is correct
yes was assigned	#tp	#fp
no was assigned	#fn	#tn

One may then calculate averages for P , R , and F across categories (*macro-averaging*) or by building a master contingency table of all the decisions made and averaging across every decision (*micro-averaging*). In micro-averaging, the largest categories will dominate, while in macro-averaging, a better picture of categorizer effectiveness is given across all the categories (although one may naturally be principally concerned with performance on the largest category sizes). Neither choice of averaging is alone sufficient;¹⁵ both convey information about the categorizer (and consequently the nature of the dataset).

Remember that categories are assigned as labels only if their category score is above some score threshold. If the threshold is taken to be very high then only neighbors which are very near or a preponderance of near like-labeled neighbors will cause the category score to exceed the score threshold; in this case, we naturally expect the precision to be very high and the recall to be very low (ie, the percentage of documents categorized properly of those categorized is high, although only a small percentage of documents in the entire collection were categorized at all). Alternatively, we may take the threshold to be very low, in which case all documents will (for large k) have all labels, and so the recall is very high while the precision is very low.

Figure 5a below is data taken from a preliminary run of the kNN categorizer on a set of 50,000 test segments from the MALACH scratchpad dataset. The categorizer was trained on 100,000 documents and used the nearest 10 neighbors to assign labels. The plot is produced by sweeping the score threshold across the score range (where, again, highest threshold is at highest precision and lowest threshold is at highest recall) and, for each threshold value, computing the micro and macro-averaged precision and recall. Similarly, the F measure may be calculated across the range of thresholds. The F measure vs. threshold from the same run is plotted in Figure 5b.

The MALACH team at IBM and Craig Murray at UMD have produced some data using independent kNN and ME categorizers which the above results will be compared against. Initial indications are that the scheme here described is doing a better job with macro-averaging than micro-averaging, while this is not the case with the other kNN implementations (once the cause of this is determined, it is unavoidable that we will have learned something about the dataset—that is, we will be more able to produce an optimal implementation of kNN for the MALACH dataset, as well as better prepared to utilize the hierarchical structure on the forthcoming hierarchical probabilistic categorizer).

5 Future work

Insight from the kNN implementation will help us to develop an understanding dataset: Several important questions to be answered are,

- why should we expect different weighting/ranking schemes to perform better or worse?
- how can learning about the shortcoming of kNN imply that a hierarchical model might be an improvement?
- why are certain category types easier or harder to classify and why?

¹⁴In other words, the categorizer attempts to answer the question, does this test document belong in the category I am currently considering?

¹⁵Suppose for a labelled collection that the labels are sorted in decreasing order of frequency f , and their position in the sort is denoted by r . Zipf's law states that $f \propto \frac{1}{r}$. Roughly restated, in any collection there will be a few categories with very many documents, and many categories with only very few or only one document. These *singletons* represents a significant problem in categorization as there will typically be little or no training data available for them.

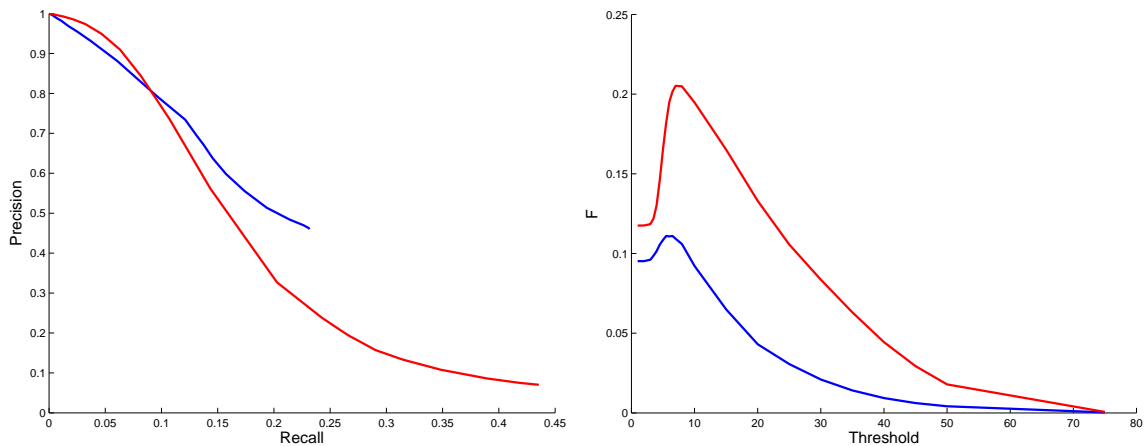


Figure 5: a. Precision-Recall curve plotted as a function of threshold. Data was taken using 10-NN, with a training and testing document sizes of 100k and 50k MALACH scratchpad segments respectively. The blue line (shorter) uses macro-averaging, while the red (longer line) is micro-averaged. b. F vs. threshold. Precision and recall are equally weighted ($\alpha = .5$). Data is from the same run. Here, the blue line (bottom curve) uses macro-averaging, while the red (top curve) is micro-averaged.

Data analysis from the kNN implementation here described and that of Craig Murray is just now beginning. The immediate next goal is to begin implementing the HPLC, which will commence over the winter term.

6 Endnotes

The project homepage is available online at [9].

References

- [1] Malach home page, JHU. <http://www.clsp.jhu.edu/research/malach/>.
- [2] D. B. Rubin A. P. Dempster, N. M. Laird. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society Series B*, vol. 39, no. 1:1–38, Nov. 1977.
- [3] H. Schütze C.D. Manning. *Foundations of Statistical Natural Language Processing*, pages 540–544, 575–578, 604–606. MIT Press, 1999.
- [4] Fabio Crestani, Mark Girolami, and C. J. van Rijsbergen, editors. *Advances in Information Retrieval, 24th BCS-IRSG European Colloquium on IR Research Glasgow, UK, March 25-27, 2002 Proceedings*, volume 2291 of *Lecture Notes in Computer Science*. Springer, 2002.
- [5] G.C. Fox K. Rose, E. Gurewitz. Statistical mechanics and phase transitions in clustering. *Physical Review Letters*, 1990.
- [6] Dibyendu Majumdar. YASE project homepage. http://www.mazumdar.demon.co.uk/yase_index.html.
- [7] R. Nakano N. Ueda. Deterministic annealing variant of the EM algorithm. *Advances in Neural Information Processing Systems 7*, pages 545–552, 1995.

- [8] Doug Oard. Malach home page, UMD. <http://raven.umd.edu/dlrg/malach/>.
- [9] J. Scott Olsson. Project home page. <http://www.math.umd.edu/~olsson/amsc663/>.
- [10] R. Baeza-Yates W.B. Frakes, editor. *Informaion Retrieval, Data Structures and Algorithms*, pages 28–36. Prentice Hall, 1992.
- [11] M.A. Weiss. *Data Structures and Algorithm Analysis in C*, pages 110–122. Addison-Wesley, 1997.