Vector-based Natural Language Call Routing

Jennifer Chu-Carroll*
Lucent Technologies Bell Laboratories

Bob Carpenter†
Lucent Technologies Bell Laboratories

This paper describes a domain-independent, automatically trained natural language call router for directing incoming calls in a call center. Our call router directs customer calls based on their response to an open-ended How may I direct your call? *prompt. Routing behavior is trained from a corpus of transcribed and hand-routed calls and then carried out using vector-based information retrieval techniques. Terms consist of* n*-gram sequences of morphologically reduced content words, while documents representing routing destinations consist of weighted term frequencies derived from calls to that destination in the training corpus. Based on the statistical discriminating power of the* n*-gram terms extracted from the caller's request, the caller is 1) routed to the appropriate destination, 2) transferred to a human operator, or 3) asked a disambiguation question. In the last case, the system dynamically generates queries tailored to the caller's request and the destinations with which it is consistent, based on our extension of the vector model. Evaluation of the call router performance over a financial services call center using both accurate transcriptions of calls and fairly noisy speech recognizer output demonstrated robustness in the face of speech recognition errors. More specifically, using accurate transcriptions of speech input, our system correctly routed 93.8% of the calls after redirecting 10.2% of all calls to a human operator. Using speech recognizer output with a 23% error rate reduced the number of correctly routed calls by 4%.*

1. Introduction

The **call routing** task is one of directing a customer's call to an appropriate destination within a call center or directly providing some simple information, such as current loan rates, on the basis of some kind of interaction with the customer. In current systems, such interaction is typically carried out via a touch-tone system with a rigid predetermined navigational menu. The primary disadvantages of navigating menus for users are the time it takes to listen to all the options and the difficulty of matching their goals to the given options. These problems are compounded by the necessity of descending a nested hierarchy of choices to zero in on a particular activity. Even requests with simple English phrasings such as *I want the balance on my car loan* may require users to navigate as many as four or five nested menus with four or five options each. We describe an alternative to touch-tone menus that allows users to interact with a call router in natural spoken English dialogues just as they would with a human operator.

In a typical dialogue between a caller and a human operator, the operator responds to a caller request by either routing the call to an appropriate destination, or querying the caller for further information to determine where the call should be routed. Thus,

* 600 Mountain Avenue, Murray Hill, NJ 07974. E-mail: jencc@research.bell-labs.com
† 600 Mountain Avenue, Murray Hill, NJ 07974. E-mail: carp@research.bell-labs.com

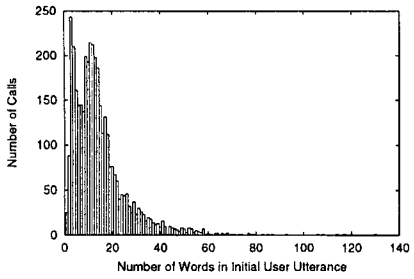
in developing an automatic call router, we select between these two options as well as a third option of sending the call to a human operator in situations where the router recognizes that to automatically handle the call is beyond its capabilities. The rest of this paper provides both a description and an evaluation of an automatic call router that consists of 1) a routing module driven by a novel application of vector-based information retrieval techniques, and 2) a disambiguation query generation module that utilizes the same vector representations as the routing module and dynamically generates queries tailored to the caller's request and the destinations with which it is consistent, based on our extension of the vector model. The overall call routing system has the following desirable characteristics: First, the training of the call router is domain independent and fully automatic,[1] allowing the system to be easily ported to new domains. Second, the disambiguation module dynamically generates queries based on caller requests and candidate destinations, allowing the system to tailor queries to specific circumstances. Third, the system is highly robust to speech recognition errors. Finally, the overall performance of the system is high, in particular when using noisy speech recognizer output. With transcription (perfect recognition), we redirect 10.2% of the calls to the operator, correctly routing 93.8% of the remainder either with or without disambiguation. With spoken input processed automatically with recognition performance at a 23% word error rate, the percentage of correctly routed calls drops by only 4%.

2. Related Work

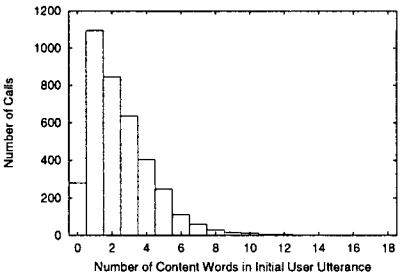
Call routing is similar to text categorization in identifying which one of n topics (or in the case of call routing, destinations) most closely matches a caller's request. Call routing is distinguished from text categorization by requiring a single destination to be selected, but allowing a request to be refined in an interactive dialogue. The closest previous work to ours is Ittner, Lewis, and Ahn (1995), in which noisy documents produced by optical character recognition are classified against multiple categories. We are further interested in carrying out the routing process using natural, conversational language.

The only work on natural language call routing to date that we are aware of is that by Gorin and his colleagues (Gorin, Riccardi, and Wright 1997; Abella and Gorin 1997; Riccardi and Gorin 1998), who designed an automated system to route calls to AT&T operators. They select salient phrase fragments from caller requests (in response to the system's prompt of *How may I help you?*), such as *made a long distance* and *the area code for*, and sometimes including phrases that are not meaningful syntactic or semantic units, such as *it on my credit*. These salient phrase fragments, which are incorporated into their finite-state language model for their speech recognizer, are then used to compute likely destinations, which they refer to as **call types**. This is done by either computing a posteriori probabilities for all possible call types (Gorin 1996) or by passing the weighted fragments through a neural network classifier (Wright, Gorin, and Riccardi 1997). Abella and Gorin (1997) utilized the Boolean formula minimization algorithm for combining the resulting set of call types based on a hand-coded hierarchy of call types. This algorithm provides the basis for determining whether or not the goal of the request can be uniquely identified, in order to select from a set of dialogue strategies for response generation.

1 The training process is automatic except for minor editing of a standard stop list, which will be discussed in Section 4.1.2, and for the mapping between n-gram morphologically reduced noun phrases and their expanded forms in the disambiguation process, which will be discussed in Section 4.2.



(a) Total words



(b) Content words

Figure 1
Histogram of call lengths.

3. Corpus Analysis

To examine human-human dialogue behavior, we analyzed a set of 4,497 transcribed telephone calls involving actual customers interacting with human call operators at a large call center. In the vast majority of these calls, the first customer utterance contained between 1 and 20 words, while the longest first utterance had 131 words. However, these utterances included only a few **content words**,[2] with almost all calls containing fewer than 10 content words in the initial user utterance. Figures 1(a) and 1(b) show histograms of call lengths based on total words and content words in the initial user utterance in each call, respectively.

Figure 2 shows the distribution of calls to the top 23 destinations on a log scale in our corpus.[3] The perplexity of a probability distribution provides a measure of the difficulty of classification of samples drawn from that distribution. Using the estimate of call distribution based on Figure 2, our task perplexity is 6.836.[4]

We further analyzed our corpus of calls along two dimensions: the semantics of caller requests and the dialogue actions for operator responses. The analysis of the semantics of caller requests is intended to examine the ways in which users typically express their goal when prompted, and is used to focus on an appropriate subset of the classes of user utterances that the call router should handle automatically (as opposed to transferring to a human operator). The analysis of the dialogue actions for operator responses, on the other hand, is intended to determine the types of responses the call router should be able to provide in response to user utterances in order to help design the response generation component of the call router. The analysis of the corpus along both dimensions was performed by the first author.

3.1 Semantics of Caller Requests

In our corpus, all callers respond to an initial open-ended prompt of ⟨ABC⟩ *banking services call director; how may I direct your call?* Their responses varied greatly in their

2 Content words are keywords automatically extracted from the training corpus that are considered relevant for routing purposes. For details on how the list of content words is selected, see Section 4.1.2.

3 These are destinations that received more than 10 calls in the corpus we analyzed.

4 Recall that the **entropy** of a distribution p is the expected value of the log probability, given by $H(p) = -\sum_x p(x) \log_2 p(x)$. The **perplexity** is given by $2^{H(p)}$ and can be thought of roughly as the number of equiprobable categories that would lead to the same classification difficulty.

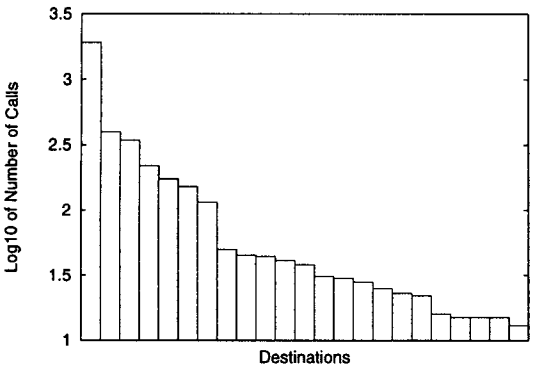


Figure 2
Distribution of calls.

degree of specificity. We roughly classified the calls into the following three broad classes:

> **Destination Name,** in which the caller explicitly specifies the name of the department to which he wishes to be transferred. The requested destination can form an answer to the operator's prompt by itself, as in *deposit services*, or be part of a complete sentence, as in *I would like to speak to someone in auto leasing please.*
>
> **Activity,** in which the caller provides a description of the activity he wishes to perform, and expects the operator to transfer his call to the appropriate department that handles the given activity. Such descriptions may be ambiguous or unambiguous, depending on the level of detail the caller provides, which in turn depends on the caller's understanding of the organization of the call center. Because all transactions related to savings accounts are handled by the deposit services department in the call center we studied, the request *I want to talk to someone about savings accounts* will be routed to Deposit Services. On the other hand, the similar request *I want to talk to someone about car loans* is ambiguous between Consumer Lending, which handles new car loans, and Loan Services, which handles existing car loans. Queries can also be ambiguous due to the caller's providing more than one activity, as in *I need to get my checking account balance and then pay a car loan.*
>
> **Indirect Request,** in which the caller describes his goal in a roundabout way, often including irrelevant information. This typically occurs with callers who are unfamiliar with the call center organization, or those who have difficulty concisely describing their goals. An example of an actual indirect request is *ah I'm calling 'cuz ah a friend gave me this number and ah she told me ah with this number I can buy some cars or whatever but she didn't know how to explain it to me so I just called you you know to get that information.*

Table 1 shows the distribution of caller requests in our corpus with respect to these semantic types. Our analysis shows that in the vast majority of calls, the request was based on either destination name or activity. Since in our corpus there are only 23 dis-

Table 1
Semantic types of caller requests.

	Destination Name	Activity	Indirect Request
# of calls	949	3271	277
% of all calls	21.1%	72.7%	6.2%

Table 2
Call operator dialogue actions.

	Notification	Query	
		NP	Others
# of calls	3,608	657	232
% of all calls	80.2%	14.6%	5.2%

tinct destinations,[5] and each destination only handles a fairly small number (dozens to hundreds) of activities, requests based on destination names and activities are expected to be more predictable and thus more suitable for handling by an automatic call router. However, our system does not directly classify calls in terms of specificity; this classification was only intended to provide a sense of the distribution of calls received.

3.2 Dialogue Actions for Operator Responses

In addition to analyzing how the callers phrased their requests in response to the operator's initial prompt, we also analyzed how the operators responded to the callers' requests.[6] We found that in our corpus, the human operator either **notifies** the caller of a destination to which the call will be transferred, or **queries** the caller for further information, most frequently when the original request was ambiguous and, much less often, when the original request was not heard or understood.

Table 2 shows the frequency with which each dialogue action was employed by human operators in our corpus. It shows that nearly 20% of all caller requests require further disambiguation. We further analyzed these calls that were not immediately routed and noted that 75% of them involve underspecified noun phrases, such as requesting *car loans* without specifying whether it is an existing car loan or a new car loan. The remaining 25% mostly involve underspecified verb phrases, such as asking to *transfer funds* without specifying the accounts to and from which the transfer will take place, or missing verb phrases, such as asking for *direct deposit* without specifying whether the caller wants to set up a direct deposit or change an existing direct deposit.

Based on our analysis of operator responses, we decided to first focus our router responses on notifying the caller of a selected destination in cases where the caller request is unambiguous, and on formulating a query for noun phrase disambiguation in the case of noun phrase underspecification in the caller request. For calls that

5 Although the call center had nearly 100 departments, in our corpus of 4,500 calls, only 23 departments received more than 10 calls. We chose to base our experiments on these 23 destinations.

6 In most calls, we analyzed the utterances given in the operator's second turn in the dialogue. However, in situations where the operator generates an acknowledgment, such as *uh-huh*, midway through the caller's request, we analyzed utterances in the next operator turn.

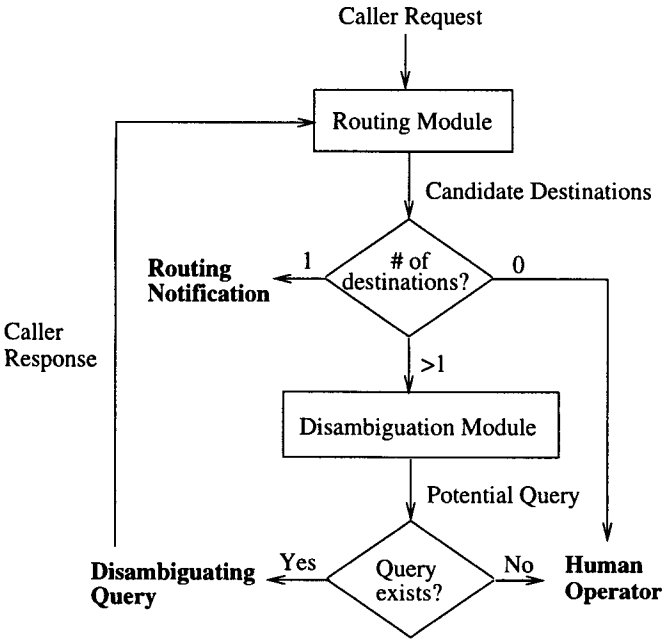


Figure 3
Call router architecture.

do not satisfy either criterion, the call router should simply relay them to a human operator.[7]

4. Vector-based Call Routing

In addition to notifying the caller of a selected destination or querying the caller for further information, an automatic call router should be able to identify when it is unable to handle a call and route the call to a human operator for further processing. The process of determining whether to route a call, generate a disambiguation query, or redirect the call to an operator is carried out by two modules in our system, the **routing module** and the **disambiguation module**, as shown in Figure 3. Given a caller request, the routing module selects a set of candidate destinations to which it believes the call can reasonably be routed. If there is exactly one such destination, the call is routed to that destination and the caller notified; if there is no appropriate destination, the call is sent to an operator; and if there are multiple candidate destinations, the disambiguation module is invoked. In the last case, the disambiguation module attempts to formulate a query that it believes will solicit relevant information from the caller to allow the revised request to be routed to a unique destination. If such a query is successfully formulated, it is posed to the caller, and the system makes another attempt at routing the revised request, which includes the original request and the caller's response to the follow-up question; otherwise, the call is sent to a human operator.

7 Note that the corpus analysis described in this section was conducted with the purpose of determining guidelines for system design in order to achieve reasonable coverage of phenomena in actual human-human dialogues. The call classification schemes presented in this section do not come into play in the actual training or testing of our system, nor do we discard any part of our training corpus as a result of this analysis.

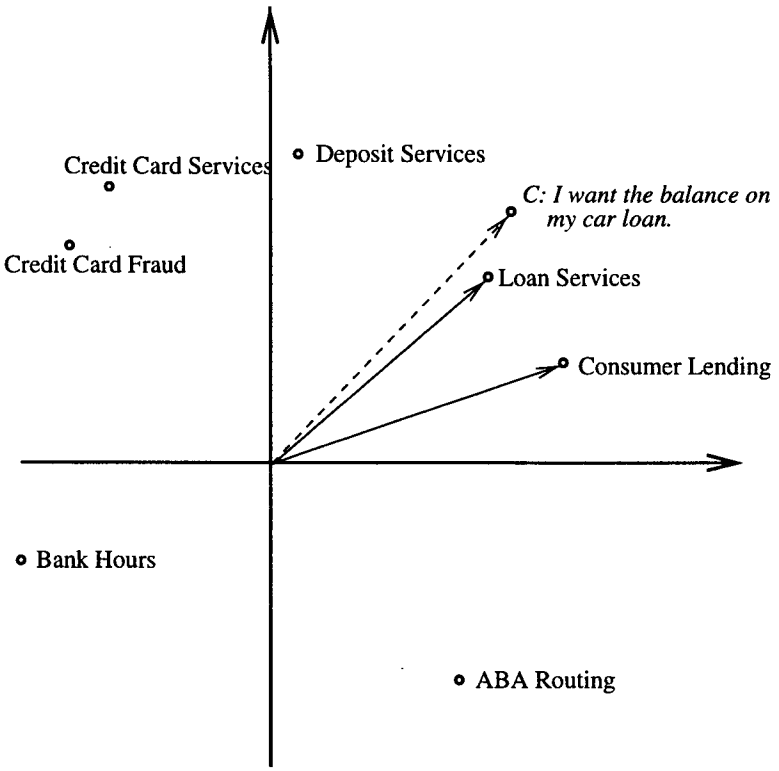


Figure 4
Two-dimensional vector representation for the routing module.

Our approach to call routing is novel in its application of vector-based information retrieval techniques to the routing process, and in its extension of the vector-based representation for dynamically generating disambiguation queries (Chu-Carroll and Carpenter 1998). The routing and disambiguation mechanisms are detailed in the following sections.

4.1 The Routing Module

4.1.1 Vector Representation for the Routing Module. In vector-based information retrieval, the database contains a large collection of documents, each of which is represented as a vector in n-dimensional space. Given a query, a query vector is computed and compared to the existing document vectors, and those documents whose vectors are **similar** to the query vector are returned. We apply this technique to call routing by treating each destination as a document, and representing the destination as a vector in n-dimensional space. Given a caller request, an n-dimensional request vector is computed. The similarity between the request vector and each destination vector is then computed and those destinations that are close to the request vector are then selected as the candidate destinations. This vector representation for destinations and query is illustrated in a simplified two-dimensional space in Figure 4.

In order to carry out call routing with the aforementioned vector representation, three issues must be addressed. First, we must determine the vector representation for each destination within the call center. Once computed, these destination vectors should remain constant as long as the organization of the call center remains

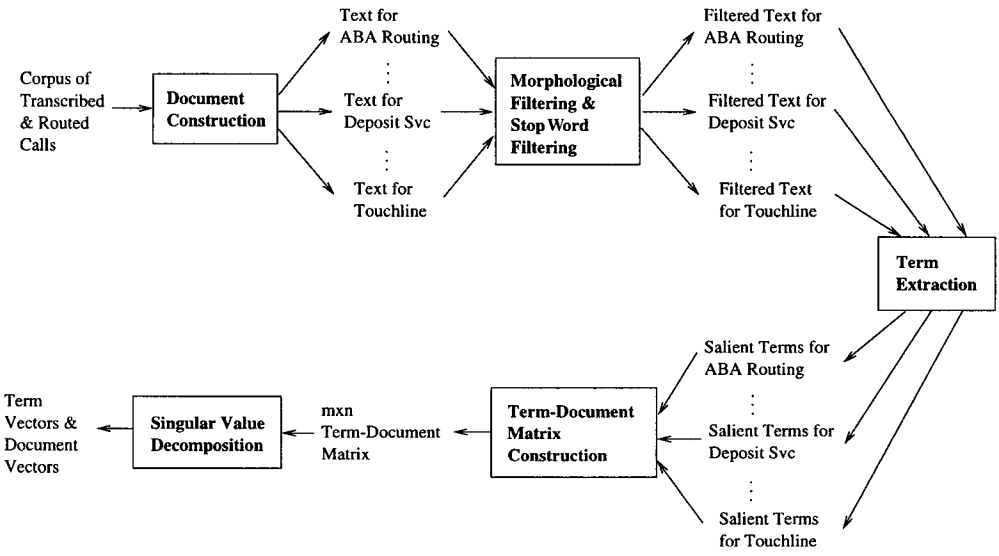


Figure 5
Training process for the routing module.

unchanged.[8] Second, we must determine how a caller request will be mapped to the same vector space for comparison with the destination vectors. Finally, we must decide how the similarity between the request vector and each destination vector will be measured in order to select candidate destinations.

4.1.2 The Training Process. The goal of the training phase of the call router is to determine the values of the destination vectors (and term vectors) that will subsequently be used in the routing process. Our training process, depicted in Figure 5, requires a corpus of transcribed calls, each of which is routed to the appropriate destination.[9] These routed calls are processed by five domain-independent procedures to obtain the desired document (destination) and term vectors.

Document Construction. Since our goal is to represent each destination as an n-dimensional vector, we must create one (virtual) document per destination. The document for a destination contains the raw text of the callers' contributions in all calls routed to that destination, since these are the utterances that provided vital information for routing purposes. For instance, the document for deposit services may contain utterances such as *I want to check the balance in my checking account* and *I would like to stop payment on a check*. In our experiments, the corpus contains 3,753 calls routed to 23 destinations.[10]

8 One may consider allowing the call router to constantly update the destination vectors as new data are being collected while the system is deployed. We leave adding learning capabilities to the call router for future work.

9 The transcription process can be carried out by humans or by an automatic speech recognizer. In the experiments reported in this paper, we used human transcriptions.

10 These calls are a subset of the 4,500 calls used in our corpus analysis. We included calls of all semantic types, but excluded calls to destinations that were not represented by more than 10 calls, as well as ambiguous calls that were not resolved by the operator.

Morphological Filtering and Stop Word Filtering. For routing purposes, we are concerned with the semantics of the words present in a document, but not with the morphological forms of the words themselves. Thus we filter each (virtual) document, produced by the document construction process, through the morphological processor of the Bell Labs Text-to-Speech synthesizer (Sproat 1998) to extract the root form of each word in the corpus. This process will reduce singulars, plurals, and gerunds to their root forms, such as reducing *service, services,* and *servicing* to the root *service.* Also, the various verb forms are also reduced to their root forms, such as reducing *going, went,* and *gone* to *go.*[11]

Next, the root forms of caller utterances are filtered through two lists, the **ignore list** and the **stop list,** in order to build more accurate *n*-gram term models for subsequent processing. The ignore list consists of noise words, which are common in spontaneous speech and can be removed without altering the meaning of an utterance, such as *um* and *uh.* These words sometimes get in the way of proper *n*-gram extraction, as in *I'd like to speak to someone about a car uh loan.* When the noise word *uh* is filtered out of the utterance, we can then properly extract the bigram *car+loan.* The stop list enumerates words that are ubiquitous and therefore do not contribute to discriminating between destinations, such as *the, be, for,* and *morning.* We modified the standard stop list distributed with the SMART information retrieval system (Salton 1971) to include domain-specific terms and proper names that occurred in our training corpus.[12] Note that when a word on the ignore list is removed from an utterance, it allows words preceding and succeeding the removed word to form *n*-grams, such as *car+loan* in the example above. On the other hand, when a stop word is removed from an utterance, a placeholder is inserted into the utterance to prevent the words preceding and following the removed stop word from forming *n*-grams. For instance, after stop word filtering, the caller utterance *I want to check on an account* becomes ⟨*sw*⟩ ⟨*sw*⟩ ⟨*sw*⟩ *check* ⟨*sw*⟩ ⟨*sw*⟩ *account,* resulting in the two unigrams *check* and *account.* Without the placeholders, we would extract the bigram *check+account,* just as if the caller had used the term *checking account* in the utterance.

In our experiments, the ignore list contains 25 words, which are variations of common transcriptions of speech disfluencies, such as *ah, aah,* and *ahh.* The stop list contains over 1,200 words, including function words, proper names, greetings, etc.

Term Extraction. The output of the filtering processes is a set of documents, one for each destination, containing the root forms of the content words extracted from the raw texts originally in each document. In order to capture word co-occurrence, *n*-gram terms are extracted from the filtered texts. First, a list of *n*-gram terms and their counts are generated from all filtered texts. Thresholds are then applied to the *n*-gram counts to select as salient terms those *n*-gram terms that occurred sufficiently frequently. Next, these salient terms are used to reduce the filtered text for each document to a bag of salient terms, i.e., a collection of *n*-gram terms along with their respective counts. Note that when an *n*-gram term is extracted, all of the lower order *k*-grams, where $1 \leq k < n$, are also extracted. For instance, the word sequence *checking account balance* will result in the trigram *check+account+balance,* as well as the bigrams *check+account* and *account+balance* and the unigrams *check, account,* and *balance.*

11 Not surprisingly, confusion among morphological variants was a source of substantial error from the recognizer. Details can be found in Reichl et al. (1998).

12 The idea of a standard stop list in the information retrieval literature is to eliminate terms that do not contribute to discriminating among documents. We extend this notion to our application to include additional proper names such as *Alaska* and *Houston,* as well as domain- or application-specific terms such as *bye* and *cadet.* The modification to the standard stop list is performed by manually examining the unigram terms extracted from the training corpus.

In our experiments, we selected as salient terms unigrams that occurred at least twice and bigrams and trigrams that occurred at least three times. This resulted in 62 trigrams, 275 bigrams, and 420 unigrams. In our training corpus, no four-gram occurred three times. Manual examination of these n-gram terms indicates that almost all of the selected salient terms are relevant for routing purposes.[13]

Term-Document Matrix Construction. Once the bag of salient terms for each destination is constructed, it is very straightforward to construct an $m \times n$ term-document frequency matrix A, where m is the number of salient terms, n is the number of destinations, and an element $A_{t,d}$ represents the number of times the term t occurred in calls to destination d. This number indicates the degree of association between term t and destination d, and our underlying assumption is that if a term occurred frequently in calls to a destination in our training corpus, then occurrence of that term in a caller's request indicates that the call should be routed to that destination.

In the term-document frequency matrix A, a row A_t is an n-dimensional vector representing the term t, while a column A_d is an m-dimensional vector representing the destination d. However, by using the raw frequency counts as the elements of the matrix, more weight is given to terms that occurred more often in the training corpus than to those that occurred less frequently. For instance, a unigram term such as *account*, which occurs frequently in calls to multiple destinations will have greater frequency counts than say, the trigram term *social+security+number*. As a result, when the two vectors representing *account* and *social+security+number* are combined, as will be done in the routing process, the term vector for *account* contributes more to the combined vector than that for *social+security+number*. In order to balance the contribution of each term, the term-document frequency matrix is normalized so that each term vector is of unit length (later weightings do not preserve this normalization, though). Let B be the result of normalizing the term-document frequency matrix, whose elements are given as follows:

$$B_{t,d} = \frac{A_{t,d}}{(\sum_{1 \le e \le n} A_{t,e}^2)^{1/2}}$$

Our second weighting is based on the notion that a term that only occurs in a few documents is more important in routing than a term that occurs in many documents. For instance, the term *stop+payment*, which occurred only in calls to deposit services, should be more important in discriminating among destinations than *check*, which occurred in many destinations. Thus, we adopted the **inverse-document frequency (IDF)** weighting scheme (Sparck Jones 1972) whereby a term is weighted inversely to the number of documents in which it occurs. This score is given by:

$$IDF(t) = \log_2 \frac{n}{d(t)}$$

where t is a term, n is the number of documents in the corpus, and $d(t)$ is the number of documents containing the term t. If t only occurred in one document, $IDF(t) = \log_2 n$; if t occurred in every document, $IDF(t) = \log_2 1 = 0$. Thus, using this weighting scheme, terms that occur in every document will be eliminated.[14] We weight the matrix B by

13 It would have been possible to hand-edit the set of n-gram terms at this point to remove unwanted terms. The results we report in this paper use the automatically selected terms without any hand-editing.

14 To preserve all terms, we could have used a common variant of the IDF weighting where $IDF(t) = log_2 \frac{n+\epsilon}{d(t)}$ for some nonnegative ϵ.

multiplying each row t by $IDF(t)$ to arrive at the matrix C:

$$C_{t,d} = IDF(t) \cdot B_{t,d}$$

Singular Value Decomposition and Vector Representation. In the weighted term-document frequency matrix C, terms are represented as n-dimensional vectors (in our system, $n = 23$), and destinations are represented as m-dimensional vectors (in our system, $m = 757$). In order to provide a uniform representation of term and document vectors and to reduce the dimensionality of the document vectors, we applied the singular value decomposition to the $m \times n$ matrix C (Deerwester et al. 1990) to obtain:[15]

$$C = U \cdot S \cdot V^{T},$$

where

1. U is an $m \times m$ orthonormal matrix;
2. V is an $n \times n$ orthonormal matrix; and
3. S is an $m \times n$ positive matrix whose nonzero values are $s_{1,1}, \ldots, s_{r,r}$, where r is the rank of C, and they are arranged in descending order $s_{1,1} \geq s_{2,2} \geq \cdots \geq s_{r,r} > 0$.

Figure 6 illustrates the results of singular value decomposition according to the above equation. The shaded portions of the matrices are what we use as the basis for our term and document vector representations, as follows:

1. U_r is an $m \times r$ matrix, in which each row forms the basis of our term vector representation;
2. V_r is an $n \times r$ matrix, in which each row forms the basis of our document vector representation; and
3. S_r is an $r \times r$ positive diagonal matrix whose values are used for appropriate scaling in the term and document vector representations.

The actual representations of the term and document vectors are U_r and V_r scaled (or not) by elements in S_r, depending on whether the representation is intended for comparisons between terms, between documents, or between a term and a document. For instance, since the similarity between two documents can be measured by the dot product between vectors representing the two documents (Salton 1971), and $C^{T} \cdot C$ contains the dot products of all pairwise column vectors in the weighted term-document frequency matrix C, the similarity between the ith and jth documents can

15 Our original intent was to apply singular value decomposition and to reduce the dimensionality of the resulting vectors below the rank of the original matrix in order to carry out latent semantic indexing (Deerwester et al. 1990). Benefits cited for latent semantic indexing include the clustering of "synonyms" leading to improved recall. In the end, dimensionality reduction degraded performance for our data. However, our method is not equivalent to the standard approach in vector-based information retrieval, which simply uses the rows or columns of the term-document matrix (see Salton [1971] for definitions of the standard case). The difference arises through the cosine measure of similarity, which is operating over different spaces in the standard vector case and with the result of SVD in our case. Although we did not run the experiments, we believe similar results would be obtained by using cosine to compare rows or columns of the term-document matrix directly.

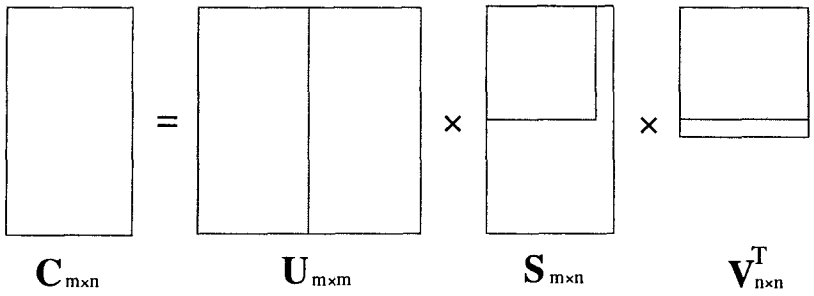


Figure 6
Singular value decomposition.

simply be recovered by element $(C^T \cdot C)_{ij}$. Since U is orthonormal, S is a diagonal matrix, we have:

$$\begin{aligned} C^T \cdot C &= (U \cdot S \cdot V^T)^T \cdot (U \cdot S \cdot V^T) \\ &= V \cdot S^T \cdot U^T \cdot U \cdot S \cdot V^T \\ &= V \cdot S \cdot S \cdot V^T \\ &= (V \cdot S) \cdot (V \cdot S)^T \end{aligned}$$

Because only the first r diagonal elements of S are nonzero, we have:

$$(V \cdot S) \cdot (V \cdot S)^T = (V_r \cdot S_r) \cdot (V_r \cdot S_r)^T$$

The above equations suggest that scaling the vectors V_r with elements in S_r, i.e., representing documents as row vectors in $V_r \cdot S_r$, facilitates comparisons between documents. The same reasoning holds for representing terms as row vectors in $U_r \cdot S_r$ for comparisons between terms, although in this particular application, we are not interested in term-term comparisons.

To measure the degree of association between a term and a document, we look up an element in the weighted term-document frequency matrix. Because S is a diagonal matrix with only the first r elements nonzero, we have:

$$\begin{aligned} C &= U \cdot S \cdot V^T \\ &= U \cdot (V \cdot S)^T \\ &= U_r \cdot (V_r \cdot S_r)^T \end{aligned}$$

Therefore, representing terms simply by row vectors in U_r and documents by row vectors in $V_r \cdot S_r$ allows us to make comparisons between documents, as well as between terms and documents.

4.1.3 Call Routing. As discussed earlier, two subprocesses need to be carried out during the call routing process. First, a **pseudodocument vector** must be constructed to represent the caller's request in order to facilitate the comparisons between the request and each document vector. Second, a method for comparison must be established to measure the similarity between the pseudodocument vector and the document vectors in $V_r \cdot S_r$, and a threshold must be determined to allow for the selection of candidate destinations.

Pseudodocument Generation. Given a caller utterance (either in text form from a keyboard interface or as the output from an automatic speech recognizer), we first perform the morphological and stop word filtering and the term extraction procedures as in the training process to extract the relevant n-gram terms from the utterance. Since higher-level n-gram terms are, in general, better indicators of potential destinations, we further allow trigrams to contribute more to constructing the pseudodocument than bigrams, which in turn contribute more than unigrams. Thus we assign a weight w_3 to trigrams, w_2 to bigrams, and w_1 to unigrams,[16] and each extracted n-gram term is then weighted appropriately to create a bag of terms in which each extracted n-gram term occurs w_n times. As a result, when we construct a pseudodocument from the bag of terms, we get the effect of weighting each n-gram term by w_n.

Given the extracted n-gram terms, we can present the request as an $m \times 1$ vector Q where each element Q_i in the vector represents the number of times the ith term occurred in the bag of terms. The vector Q is then added as an additional column vector in our original weighted term-document frequency vector C, as shown in Figure 7, and we want to find the new corresponding column vector in V, V_q, that represents the pseudodocument in the reduced r-dimensional space. Since U is orthonormal and S is a diagonal matrix, we can solve for V_q by setting

$$Q = U \cdot S \cdot V_q^{\mathrm{T}}$$

Because we want a representation of the query in the document space, we transpose Q, to yield:

$$Q^{\mathrm{T}} = V_q \cdot S \cdot U^{\mathrm{T}}$$

Finally, multiplying both sides on the right by U, we have:

$$\begin{aligned} Q^{\mathrm{T}} \cdot U &= V_q \cdot S \cdot U^{\mathrm{T}} \cdot U \\ &= V_q \cdot S \end{aligned}$$

Finally, note that for our query representation in the document space, we have $Q^{\mathrm{T}} \cdot U = Q^{\mathrm{T}} \cdot U_r$ and $V_q \cdot S = V_q \cdot S_r$. $V_q \cdot S_r$ is a pseudodocument representation for the caller utterance in r-dimensional space, and is scaled appropriately for comparison between documents. This vector representation is simply obtained by multiplying Q^{T} and U_r, or equivalently, summing the vector representing each term in the bag of n-gram terms.

Candidate Destination Selection. Once the pseudodocument vector representing the caller request is computed, we measure the similarity between each document vector in $V_r \cdot S_r$ and the pseudodocument vector. There are a number of ways one may measure the similarity between two vectors, such as using the cosine score between the vectors, the Euclidean distance between the vectors, the Manhattan distance between the vectors, etc. We follow the standard technique adopted in the information retrieval community and select the cosine score as the basis for our similarity measure. The cosine score between two n-dimensional vectors $\mathbf{x}$ and $\mathbf{y}$ is given as follows:

$$cos(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}^{\mathrm{T}}}{\sqrt{\sum_{1 \le i \le n} x_i^2 \cdot \sum_{1 \le i \le n} y_i^2}}$$

16 In our system, $w_1 = 1$; $w_2 = 2$; and $w_3 = 4$.

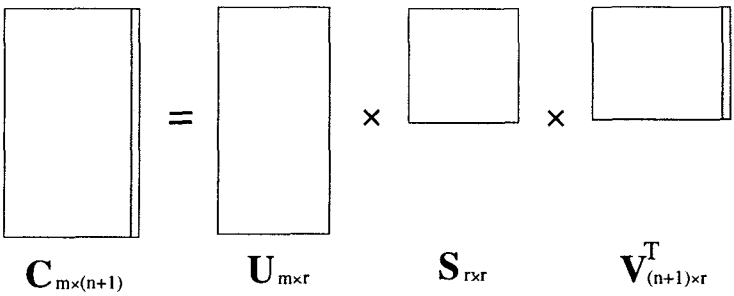


Figure 7
Pseudodocument generation.

Using cosine reduces the contribution of each vector to its angle by normalizing for length. Thus the key in maximizing cosine between two vectors is to have them point in the same direction. However, although the raw vector cosine scores give some indication of the closeness of a request to a destination, we noted that the absolute value of such closeness does not translate directly into the likelihood for correct routing. Instead, some destinations may require a higher cosine value, i.e., a closer degree of similarity, than others in order for a request to be correctly associated with those destinations. We applied the technique of logistic regression (see Lewis and Gale [1994]) in order to transform the cosine score for each destination using a sigmoid function specifically fitted for that destination. This allows us to obtain a score that represents the router's confidence that the call should be routed to that destination.

From each call in the training data, we generate, for each destination, a cosine value/routing value pair, where the cosine value is that between the destination vector and the request vector, and the routing value is 1 if the call was routed to that destination in the training data and 0 otherwise. Thus, for each destination, we have a set of cosine value/routing value pairs equal to the number of calls in the training data. The subset of these value pairs whose routing value is 1 will be equal to the number of calls routed to that destination in the training set. Then, we used least squared error to fit a sigmoid function, $1/(1+e^{-(ax+b)})$, to the set of cosine value/routing value pairs.[17] Assuming d_a and d_b are the coefficients of the fitted sigmoid function for destination d, we have the following confidence function for a destination d and cosine value x:

$$Conf(d_a, d_b, x) = 1/(1 + e^{-(d_a x + d_b)})$$

Thus the score given a request and a destination, where $\mathbf{d}$ is the vector corresponding to destination d, and $\mathbf{r}$ is the vector corresponding to the request is $Conf(d_a, d_b, \cos(\mathbf{r}, \mathbf{d}))$.

To obtain a preliminary evaluation of the effectiveness of cosine vs. confidence scores, we tested routing performance on transcriptions of 307 unseen unambiguous requests. In each case, we selected the destination with the highest cosine/confidence score to be the target destination. Using raw cosine scores, 92.2% of the calls are routed to the correct destination. On the other hand, using sigmoid confidence fitting, 93.5% of the calls are correctly routed. This yields an error reduction rate of 16.7%, illustrating the advantage of transforming the raw cosine scores to more uniform confidence scores that allow for more accurate comparisons between destinations.

17 Maximum likelihood fitting is often used rather than least squared error in logistic regression.

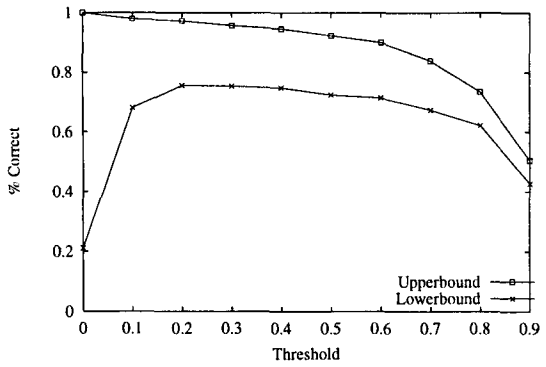


Figure 8
Router performance vs. confidence threshold.

We can compute the kappa statistic for the pure routing component of our system using the accuracies given above.[18] Recall that kappa is defined by $(a - e)/(1 - e)$ where a is the system's accuracy and e is the expected agreement by chance. Selecting destinations from the prior distribution and guessing destinations using the same distribution leads to a chance performance of $e = \sum_d P(d)^2 = 0.2858$ where the summation is over all destinations d and $P(d)$ is the percentage of calls routed to destination d. The resulting kappa score is $(0.935 - 0.2858)/(1 - 0.2858) = 0.909$.

Once we have obtained a confidence value for each destination, the final step in the routing process is to compare the confidence values to a predetermined threshold and return those destinations whose confidence values are greater than the threshold as candidate destinations. To determine the optimal value for this threshold, we ran a series of experiments to compute the upper bound and lower bound of the router's performance by varying the threshold from 0 to 0.9 at 0.1 intervals. The lower bound represents the percentage of calls that are routed correctly, while the upper bound indicates that percentage of calls that have the potential to be routed correctly after disambiguation (see Section 5 for details on upper bound and lower bound measures). Figure 8 illustrates the results of this set of experiments and shows that a threshold of 0.2 yields optimal performance. Thus we adopt 0.2 as our confidence threshold for selecting candidate destinations in the rest of our discussion.

4.1.4 Call Routing Example. To illustrate the call routing process with an example, suppose the caller responds to the operator's prompt with *I am calling to apply for a new car loan*. First the caller's utterance is passed through morphological filtering to obtain the root forms of the words in the utterance, resulting in *I am call to apply for a new car loan*. Next, words on the stop list are removed and replaced with a placeholder, resulting in ⟨*sw*⟩ ⟨*sw*⟩ *call* ⟨*sw*⟩ *apply* ⟨*sw*⟩ ⟨*sw*⟩ *new car loan*. From the filtered utterance, the router extracts the salient n-gram terms to form a bag of terms as follows: *new+car+loan, new+car, car+loan, call, apply, new, car,* and *loan*. A request vector is then computed by taking the weighted sum of the term vectors representing the salient n-gram terms, and the cosine value between this request vector and each destination vector is computed. The cosine value for each destination is subsequently transformed

18 See Siegel and Castellan (1988), and Carletta (1996) for a definition or discussion of the kappa statistic, and Walker et al. (1998) for an application of the kappa statistic to dialogue evaluation.

1.	Consumer Lending	0.979
2.	Loan Services	0.260
3.	Home Loans	0.077
4.	Collateral Control	0.069
5.	Operator	0.038

(a) Cosine Scores

1.	Consumer Lending	0.913
2.	Deposit Services	0.070
3.	PC Banking	0.049
4.	Loan Services	0.035
5.	Auto Leasing	0.032

(b) Confidence Scores

Figure 9
Ranking of candidate destinations.

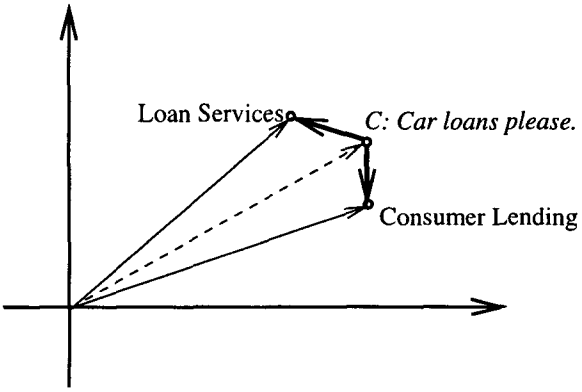


Figure 10
Two-dimensional vector representation for the disambiguation module.

using the destination-specific sigmoid function to obtain a confidence score for each destination. Figures 9(a) and 9(b) show the cosine scores and the confidence scores for the top five destinations, respectively. Given a confidence threshold of 0.2, the only candidate destination selected is Consumer Lending. Thus, the caller's request is routed unambiguously to that destination.

4.2 The Disambiguation Module

4.2.1 Vector Representation for the Disambiguation Module. When the routing module returns more than one candidate destination, the disambiguation module is invoked. The disambiguation module attempts to formulate an appropriate query to solicit further information from the caller to determine a unique destination to which the call should be routed. As discussed earlier, this occurs when two or more destination vectors are close to the request vector, as illustrated in reduced two-dimensional space in Figure 10. In the example, the caller's request *car loans please* is ambiguous since the caller does not specify whether he is interested in existing or new car loans. Therefore, the vector representation for the request falls between the vectors representing the two candidate destinations, Consumer Lending and Loan Services, and is close to both of them. The goal of the disambiguation process is to solicit an n-gram term from the caller so that when the vector representing this new n-gram term is added to the original request vector, the refined request vector will be unambiguously routed to one of the two candidate destinations. In terms of our vector representation, this means that our goal is to find term vectors that are close to the differences between the candidate destination vectors and the request vector, i.e., the highlighted vectors in Figure 10. These difference vectors, which are simply the pairwise differences of elements in each vector and are dynamically generated from the destination and request vectors, form the basis from which the disambiguation queries will be generated.

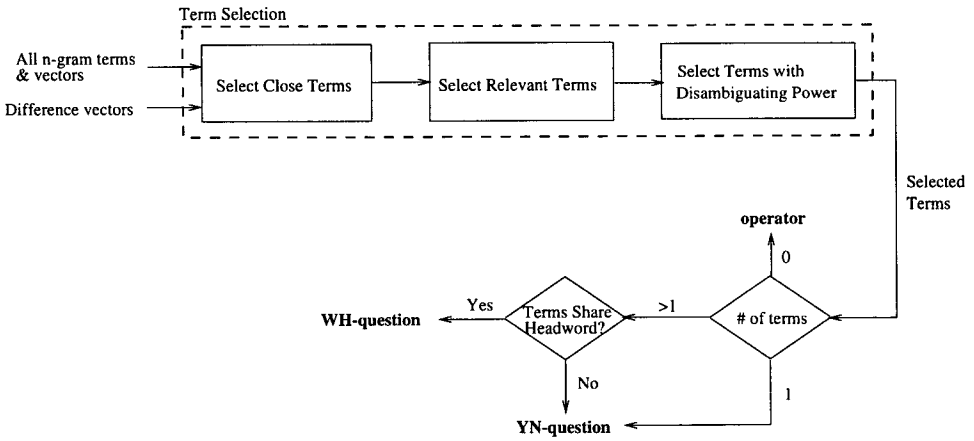


Figure 11
The disambiguation process.

4.2.2 Query Formulation. Our disambiguation module selects a subset of the salient n-gram terms from which the query will be generated. The subset of n-gram terms are those related to the original query that can likely be used to disambiguate among the candidate destinations. They are chosen by filtering all n-gram terms based on the following three criteria, as shown in Figure 11:

Closeness Since the goal of the disambiguation process is to solicit terms whose corresponding vectors are close to the difference vectors, the first step in the term selection process is to compare each n-gram term vector with the difference vectors and select those n-gram term vectors that are close to the difference vectors by the cosine measure. Since both the destination vectors and the request vector are scaled for document-document comparison in $V_r \cdot S_r$ space, the difference vectors are also represented in $V_r \cdot S_r$ space. As discussed in Section 4.1.2, documents represented in $V_r \cdot S_r$ space are suitable for comparison with terms represented in U_r space. In our system, for each difference vector, we compute the cosine score between the difference vector and each term vector, and select the 30 terms with the highest cosine scores as the set of close terms. The reasons for selecting a threshold on the number of terms instead of on the cosine score are twofold. First, in situations where many term vectors are close to the difference vector, we avoid generating an overly large set of close terms but instead focus on a smaller set of most promising terms. Second, in situations where few term vectors are close to the difference vector, we still select a set of close terms in the hope that they may contribute to formulating a reasonable query, instead of giving up on the disambiguation process outright.

Relevance From the set of close terms, we select a set of relevant terms, which are terms that further specify a term in the original request. If a term in the set of close terms can be combined with a term in the original request to form a valid n-gram term, then the resulting n-gram term is considered a relevant term. For instance, if *car+loan* is a term in the original request, then both *new* and *new+car* would produce the

relevant term *new+car+loan*. This mechanism for selecting relevant terms allows us to focus on selecting *n*-gram terms for noun phrase disambiguation by eliminating close terms that are semantically related to underspecified *n*-gram noun phrases in the original request but do not contribute to further disambiguating the noun phrases.

Disambiguating power The final criterion that we use for term selection is to restrict attention to relevant terms that can be added to the original request to result in an unambiguous routing decision using the routing mechanism described in Section 4.1.3. In other words, we augment the bag of *n*-gram terms extracted from the original request with each relevant term, and the routing module is invoked to determine if this augmented set of *n*-gram terms can be unambiguously routed to a unique destination. The set of relevant terms with disambiguating power then forms the set of selected terms from which the system's query will be formulated. If none of the relevant terms satisfy this criterion, then we include all relevant terms. Thus, instead of giving up the disambiguation process when no one term is predicted to resolve the ambiguity, the system poses a question to solicit information from the caller to move the original request one step toward being an unambiguous request. After the first disambiguation query is answered, the system subsequently selects a new set of terms from the refined, though still ambiguous, request and formulates a follow-up disambiguation query.[19]

The result of this selection process is a finite set of terms that are relevant to the original ambiguous request and, when added to it, are likely to resolve the ambiguity. The actual query is formulated based on the number of terms in this set as well as features of the selected terms. As shown in Figure 11, if the three selection criteria ruled out all *n*-gram terms, then the call is sent to a human operator for further processing. If there is only one selected term, then a *yes-no* question is formulated based on this term. If there is more than one selected term in the set, and a significant number of these terms share a common headword,[20] *X*, the system generalizes the query to ask the *wh*-question *For what type of X?* Otherwise, a *yes-no* question is formed based on the term in the selected set that occurred most frequently in the training data, based on the heuristic that a more common term is more likely to be relevant than an obscure term.[21] A third alternative would be to ask a disjunctive question, but we have not yet explored this possibility. Figure 3 shows that after the system poses its query, it attempts to route the refined request, which is the caller's original request plus the response to the disambiguation query posed by the system. In the case of *wh*-questions, *n*-gram terms are extracted from the response. For instance, if the system asks *For what type of loan?* and the user responds *It's a car loan*, then the bigram *car+loan*

19 Although it captures a similar property of each term, this criterion is computationally much more expensive than the closeness criterion. Thus, we adopt the closeness criterion to select a fixed number of candidate terms and then apply the more expensive, but more accurate, criterion to the much smaller set of candidate terms.

20 In our implemented system, this path is selected if 1) there are five or less selected terms and they all share a common headword, or 2) there are more than five terms and at least five of them share a common headword.

21 The generation of natural language queries is based on templates for *wh*-questions and *yes-no* questions. The generation process consults a manually constructed mapping between *n*-gram morphologically reduced noun phrases and their expanded forms. For instance, it maps the *n*-gram term *exist + car + loan* to *an existing car loan*. This mapping is the only manual effort needed to port the disambiguation module to a new domain.

is extracted from the response. In the case of *yes-no* questions, the system determines whether a *yes* or *no* answer is given.[22] In the case of a *yes* response, the term selected to formulate the disambiguation query is considered the caller's response, while in the case of a *no* response, the response is treated as in responses to *wh*-questions. For instance, if the user says *yes* in response to the system's query *Is this an existing car loan?*, then the trigram term *exist+car+loan* in the system's query is considered the user's response.

Note that our disambiguation mechanism, like our training process for basic routing, is domain-independent (except for the manual construction of a mapping between *n*-gram noun phrases and their expanded forms). It utilizes the set of *n*-gram terms, as well as term and document vectors that were obtained by the training of the call router. Thus, the call router can be ported to a new task with only very minor domain-specific work on the disambiguation module.

4.2.3 Disambiguation Example. To illustrate the disambiguation module of our call router, consider the request *Loans please*. This request is ambiguous because the call center we studied handles mortgage loans separately from all other types of loans, and for all other loans, existing loans and new loans are also handled by different departments.

Given this request, the call router first performs morphological, ignore word, and stop word filterings on the input, resulting in the filtered utterance of *loan* ⟨*sw*⟩. *N*-gram terms are then extracted from the filtered utterance, resulting in the unigram term *loan*. Next, the router computes a pseudodocument vector that represents the caller's request, which is compared in turn with the destination vectors. The cosine values between the request vector and each destination vector are then mapped into confidence values. Using a confidence threshold of 0.2, we have two candidate destinations, Loan Services and Consumer Lending; thus the disambiguation module is invoked.

Our disambiguation module first selects from all *n*-gram terms those whose term vectors are close to the difference vectors, i.e., the differences between each candidate destination vector and the request vector. This results in a list of 60 close terms, the vast majority of which are semantically close to *loan*, such as *auto+loan*, *payoff*, and *owe*. Next, the relevant terms are constructed from the set of close terms by selecting those close terms that form a valid *n*-gram term with *loan*. This results in a list of 27 relevant terms, including *auto+loan* and *loan+payoff*, but excluding *owe*, since neither *loan+owe* nor *owe+loan* constitutes a valid bigram. The third step is to select those relevant terms with disambiguating power, resulting in 18 disambiguating terms. Since 11 of these terms share a head noun *loan*, a *wh*-question is generated based on this headword, resulting in the query *For what type of loan?*

Suppose in response to the system's query, the user answers *Car loan*. The router then adds the new bigram *car+loan* and the two unigrams *car* and *loan* to the original request and attempts to route the refined request. This refined request is again ambiguous between Loan Services and Consumer Lending because the caller did not specify whether it was an existing or new car loan. Again, the disambiguation module selects the close, relevant, and disambiguating terms, resulting in a unique trigram *exist+car+loan*. Thus, the system generates the *yes-no* question *Is this about an existing*

22 In our current system, a response is considered a *yes* response only if it explicitly contains the word *yes*. However, as discussed in Green and Carberry, (1994) and Hockey et al. (1997), responses to *yes-no* questions may not explicitly contain a *yes* or *no* term. We leave incorporating a more sophisticated response understanding model, such as Green and Carberry (1994), into our system for future work.

car loan?[23] If the user responds *yes*, then the trigram *exist+car+loan* is added to the refined request and the call unambiguously routed to Loan Services; if the user says *No, it's a new car loan*, then the trigram *new+car+loan* is extracted from the response and the call routed to Consumer Lending.[24]

5. Evaluation of the Call Router

5.1 Routing Module Performance

We performed an evaluation of the routing module of our call router on a set of 389 calls disjoint from the training corpus. Of the 389 requests, 307 were unambiguous and routed to their correct destinations, and 82 were ambiguous and annotated with a list of potential destinations. Unfortunately, in this test set, only the caller's utterance in response to the system's initial prompt of *How may I direct your call?* was recorded and transcribed; thus we have no information about where the ambiguous calls should be routed after disambiguation. We evaluated the routing module performance on both transcriptions of caller utterances as well as output of the Bell Labs Automatic Speech Recognizer (Reichl et al. 1998) based on speech input of caller utterances (Carpenter and Chu-Carroll 1998).

5.1.1 Term Extraction Performance. Since the vector representation for caller requests is computed based on the term vectors representing the n-gram terms extracted from the requests, the performance of our call router is directly tied to the the accuracy of terms extracted from each caller utterance. Given the set of n-gram terms obtained from the training process, the accuracy of extraction of such terms based on transcriptions of caller utterances is 100%. However, when using the output of an automatic speech recognizer as input to our call router, deletions of terms present in the caller's request as well as insertions of terms that did not occur in the request affect the term extraction accuracy and thus the routing performance.

We evaluated the output of the automatic speech recognizer based on both word accuracy and term accuracy, as shown in Table 3.[25] Word accuracy is measured by taking into account all words in the transcript and in the recognized string. Two sets of results are given for word accuracy, one based on raw forms of words and the other based on comparisons of the root forms of words, i.e., after both the transcript and the recognized string are sent through the morphological filter. Term accuracy is measured by taking into account only the set of actual/recognized words that contribute to routing performance, i.e., after both the transcript and the recognized string are sent through the term extraction process.

For each evaluation dimension, we measured the recognizer performance by calculating the precision and recall. Precision is the percentage of words/terms in the recognizer output that are actually in the transcription, i.e., percentage of found words/terms

23 Recall that our current system uses simple template filling for response generation by utilizing manually constructed mappings from n-gram terms to their inflected forms, such as from *exist+car+loan* to *an existing car loan*.

24 The current implementation of the system requires that the user specify the correct answer when providing a *no* answer to a *yes-no* question, in order for the call to be properly disambiguated. However, it is possible that a system may attempt to disambiguate given a simple *no* answer by considering the n-gram term being queried (*exist+car+loan* in the above example) as a negative feature, subtracting its vector representation from the query, and attempting to route the resulting vector representation.

25 In computing the precision and recall figures, we did not take into account multiple occurrences of the same word. In other words, we consider a word in the recognized string correct if the word occurs in the transcribed text. For comparison purposes, the standard speech recognition accuracy on raw ASR output is 69.94%.

Table 3
Word accuracy vs. term accuracy on ASR output.

	Word Accuracy		Term Accuracy		
	Raw	Rooted	Unigram	Bigram	Trigram
Precision	78.6%	79.8%	93.7%	96.5%	98.5%
Recall	76.0%	77.2%	88.4%	85.5%	83.6%

that are correct, while recall is the percentage of words/terms in the transcription that are correctly returned by the recognizer, i.e., percentage of actual words/terms that are found. Table 3 shows that using the root forms of words results in a 1% absolute improvement (approximately 5% error reduction) in both precision and recall over using the raw forms of words.

A comparison of the rooted word accuracy and the unigram accuracy shows that the recognizer performs much better on content words than on all words combined. Furthermore, comparisons among term accuracies for various n-gram terms show that as n increases, precision increases while recall decreases. This is because finding a correct trigram requires that all three unigrams that make up the trigram be correctly recognized in order, hence the low recall. On the other hand, this same feature makes it less likely for the recognizer to postulate a trigram by chance, hence the high precision. An overall observation in the results presented in Table 3 is that the speech recognizer misses between 12–17% of the n-gram terms used by the call router, and introduces an extra 1–6% of n-gram terms that should not have existed. In the next section, we show how these deletions and insertions of n-gram terms affect the call router's performance.

5.1.2 Destination Selection Performance. In evaluating the performance of the routing module, we compare the list of candidate destinations with the manually annotated correct destination(s) for each call. The routing decision for each call is classified into one of eight classes, as shown in Figure 12. For instance, class **2a** contains those calls that 1) are actually unambiguous, 2) are considered ambiguous by the router, and 3) have the potential to be routed to the correct destination, i.e., the correct destination is one of the candidate destinations. On the other hand, class **3b** contains those calls that 1) are actually ambiguous, 2) are considered unambiguous by the router, and 3) are routed to a destination that is not one of the potential destinations.

We evaluated the router's performance on three subsets of our test data: unambiguous requests alone, ambiguous requests alone, and all requests combined. For each set of data, we calculated a lower bound performance, which measures the percentage of calls that are correctly routed, and an upper bound performance, which measures the percentage of calls that are either correctly routed or have the potential to be correctly routed. Table 4 shows how the upper bounds and lower bounds are computed based on the classification in Figure 12 for each of the three data sets. For instance, for unambiguous requests (classes **1** and **2**), the lower bound is the number of calls actually routed to the correct destination (class **1a**) divided by the number of total unambiguous requests, while the upper bound is the number of calls actually routed to the correct destination (class **1a**) plus the number of calls that the router finds to be ambiguous between the correct destination and some other destination(s) (class **2a**), divided by the number of unambiguous requests. The calls in **2a** are con-

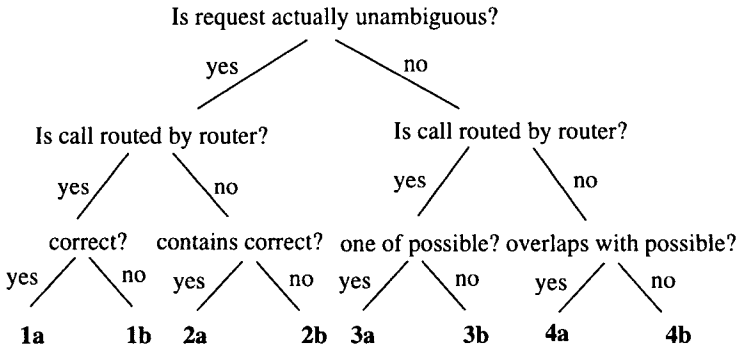


Figure 12
Classification of routing module outcome.

Table 4
Calculation of upper bounds and lower bounds.

	Unambiguous Requests	Ambiguous Requests	All Requests
Lower bound	1a/(1+2)	4a/(3+4)	(1a+4a)/all
Upper bound	(1a+2a)/(1+2)	(3a+4a)/(3+4)	(1a+2a+3a+4a)/all

Table 5
Routing results with threshold = 0.2.

	Class **a**	Class **b**
Class **1**	246	5
Class **2**	51	5
Class **3**	33	1
Class **4**	48	0

(a) Results on transcriptions

	Class **a**	Class **b**
Class **1**	239	9
Class **2**	49	10
Class **3**	30	3
Class **4**	42	7

(b) Results on ASR output

sidered potentially correct because it is likely that the call will be routed to the correct destination after disambiguation.

Tables 5(a) and 5(b) show the number of calls in our testing corpus that fell into the classes illustrated in Figure 12 based on transcriptions of caller requests and the output of an automatic speech recognizer, respectively. Tables 6(a) and 6(b) show the upper bound and lower bound performance for the three test sets based on the results in Tables 5(a) and (b), as well as the evaluation mechanism in Table 4. These results show that the system's overall performance in the case of perfect recognition falls somewhere between 75.6% and 97.2%, while the performance using our current automatic speech recognizer (ASR) output falls between 72.2% and 92.5%. The actual performance of the system is determined by two factors: 1) the performance of the disambiguation module, which determines the correct routing rate of the unambiguous calls that are considered ambiguous by the router (class **2a**, 16.6% of all unambiguous calls with transcription and 15.9% with ASR output), and 2) the percentage of calls that were routed correctly out of the ambiguous calls that were considered unambiguous by the router (class **3a**, 40.4% of all ambiguous calls with transcription and 36.6% with ASR output). Note that the performance figures given in Tables 6(a) and 6(b) are

Table 6
Router performance with threshold = 0.2.

	Unambiguous Requests	Ambiguous Requests	All Requests
Lower bound	80.1%	58.5%	75.6%
Upper bound	96.7%	98.8%	97.2%

(a) Performance on transcriptions

	Unambiguous Requests	Ambiguous Requests	All Requests
Lower bound	77.9%	51.2%	72.2%
Upper bound	93.8%	87.8%	92.5%

(b) Performance on ASR output

based on 100% automatic routing. In the next section, we discuss the performance of the disambiguation module, which determines the overall system performance, and show how allowing calls to be redirected to human operators affects the system's performance.

5.2 Disambiguation Module Performance

To evaluate our disambiguation module, we needed dialogues that satisfy two criteria. First, the caller's first utterance must be ambiguous. Second, the operator must have asked a follow-up question to disambiguate the request and have subsequently routed the call to the appropriate destination. We used 157 calls that met these two criteria as our test set for the disambiguation module. Note that this test set is disjoint from the test set used in the evaluation of the call router, since none of the calls in that set satisfied the second criterion (those calls were not recorded or transcribed beyond the caller's response to the operator's prompt). Furthermore, for this test set, we only had access to the transcriptions of the calls but not the original speech files.

For each ambiguous call, the first caller utterance was given to the router as input. The outcome of the router was classified as follows:

> **Unambiguous** if the call was routed to the selected destination. This routing was considered correct if the selected destination was the same as the actual destination and incorrect otherwise.
>
> **Ambiguous** if the router attempted to initiate disambiguation. The outcome of the routing of these calls was determined as follows:
>
> > **Correct** if a disambiguation query was generated which, when answered, led to the correct destination.[26]
> > **Incorrect** if a disambiguation query was generated which, when answered, could not lead to a correct destination.

26 Since our corpus consists of human-human dialogues, we do not have human responses to the exact disambiguation questions that our system generates. We consider a disambiguation query correct if it attempts to solicit the same type of information as the human operator, regardless of syntactic phrasing, and if answered based on the user's response to the human operator's question, led to the correct destination.

Table 7
Performance of disambiguation module on ambiguous calls.

Routed As Unambiguous		Routed As Ambiguous		
Correct	Incorrect	Correct	Incorrect	Reject
40	12	60	3	42

Reject if the router could not form a sensible query or was unable to gather sufficient information from the user after its queries and routed the call to a human operator.

Table 7 shows the number of calls that fall into each of the five categories. Out of the 157 calls, the router automatically routed 115 either with or without disambiguation (73.2%). Furthermore, 87.0% of these automatically routed calls were sent to the correct destination. Notice that out of the 52 ambiguous calls that the router considered unambiguous, 40 were routed correctly (76.9%). This is because our statistically trained call router is able to distinguish between cases where a semantically ambiguous request is equally likely to be routed to two or more destinations, and situations where the likelihood of one potential destination overwhelms that of the other(s). In the latter case, the router routes the call to the most likely destination instead of initiating disambiguation, which has been shown to be an effective strategy; not surprisingly, human operators are also prone to guess the destination based on likelihood and route calls without disambiguation.

5.3 Overall Performance

Our final evaluation of the overall performance of the call router is calculated by applying the results for evaluating the disambiguation module in Section 5.2 to the results for the routing module in Section 5.1. Tables 8(a) and 8(b) show the percentage of calls that will be correctly routed, incorrectly routed, and rejected, if we apply the performance of the disambiguation module (Table 7) to the calls that fall into each class in the evaluation of the routing module (Table 5).[27] For instance, the performance of transcribed class **2** calls (unambiguous calls that the router considered ambiguous) is computed as follows:

$$\begin{aligned} \text{Correct percentage} &= \textit{correct/total} \\ &= (51 * 60/105)/389 \\ &= 7.5\% \\ \text{Incorrect percentage} &= \textit{incorrect/total} \\ &= (5 + 51 * 3/105)/389 \\ &= 1.7\% \end{aligned}$$

27 Note that the results in Table 8(b) are only a rough upper bound for the system's overall performance on recognizer output, since the performance of the disambiguation module presented in Table 7 is evaluated on transcribed texts (because we were not able to obtain any speech data that were recorded and transcribed beyond the caller's initial response to the system's prompt). In reality, the insertions and deletions of n-gram terms in the recognizer output may lead to some inappropriate disambiguation queries or more rejections to human operators. In addition, users may provide useful information not solicited by the system's query.

Table 8
Overall performance of call router.

	Correct	Incorrect	Reject
Class 1	63.2%	1.3%	0%
Class 2	7.5%	1.7%	5.3%
Class 3	6.5%	2.2%	0%
Class 4	7.0%	0.4%	4.9%
Total	84.2%	5.6%	10.2%

(a) Performance on transcriptions

	Correct	Incorrect	Reject
Class 1	61.4%	2.3%	0%
Class 2	7.2%	2.9%	5.0%
Class 3	5.9%	2.6%	0%
Class 4	6.3%	2.1%	4.3%
Total	80.8%	9.9%	9.3%

(b) Performance on ASR output

$$\begin{aligned} \text{Rejected percentage} &= \textit{rejected/total} \\ &= (51 * 42/105)/389 \\ &= 5.3\% \end{aligned}$$

The results in Table 8(a) show that, with perfect recognition, our call router sends 84.2% of all calls in our test set to the correct destination either with or without disambiguation, sends 5.6% of all calls to the incorrect destination, and redirects 10.2% of the calls to a human operator. In other words, our system attempts to automatically handle 89.8% of the calls, of which 93.8% are routed to their correct destinations. When speech recognition errors are introduced to the routing module, the percentage of calls correctly routed decreases while that of calls incorrectly routed increases. However, it is interesting to note that the rejection rate decreases, indicating that the system attempted to handle a larger portion of calls automatically.

5.4 Performance Comparison with Existing Systems

As discussed in Section 2, Gorin and his colleagues have experimented with various methodologies for relating caller utterances with call types (destinations). Their system performance is evaluated by comparing the most likely destination returned by their call type classifier given the first caller utterance with a manually annotated list of destinations labeled based again on the first caller utterance. A call is considered correctly classified if the destination returned by their classifier is present in the list of possible destinations. In other words, their evaluation scheme is similar to our method for computing the upper bound performance of our router discussed in Section 5.1.2. We evaluated our router using their evaluation scheme with a rejection threshold of 0.2 on both transcriptions and recognition output on our original set of 389 calls used in evaluating the routing module. Table 9 shows a comparison of our system's performance and the best-performing version of their system as reported in Wright, Gorin, and Riccardi (1997); henceforth WGR97.[28]

Without other measures of task complexity, it is impossible to directly compare our results with those of WGR97. In several respects, their task is substantially different than ours. Their task is simpler in that there are fewer possible activities that a caller might request and fewer overall destinations; but it is more complex in that vocabulary

28 Wright, Gorin, and Riccardi (1997) presents system performance in the form of a rejection rate vs. correct classification rate graph, with rejection rate ranging between 10-55% and correct classification rate ranging between 63-94%. We report on two sets of results from their graph in Table 9, one with the lowest rejection rate and one that they chose to emphasize in their paper.

Table 9
Evaluation of our system and WGR97.

	# of Destinations	On Transcription		On ASR Output	
		Rejection Rate	Correct Rate	Rejection Rate	Correct Rate
Our system	23	0%	94%	3%	92%
WGR97	14	10%	84%	12%	78%
WGR97	14	40%	94%	40%	83%

items like cities are far more open-ended. Furthermore, it appears that they have many more instances of callers requesting services from more than one destination.

Comparison with human operators was not possible for our task as their routing accuracy has not been evaluated. Our transcriptions clearly indicate that they all make a substantial number of routing errors (5–10% or more), with a large degree of variation among operators.

6. Future Work

In our current system, we perform morphological reduction context-independently without regard to word class. Ideally, we would have distinguished the uses of the word *check* as a verb from its uses as a noun, requiring both training and run-time category disambiguation.

We are also interested in further clustering words that are similar in meaning, such as *car*, *auto*, and *automobile*, even though they are not related by regular morphological processes. For our application, digits or sequences of digits might be conflated into a single term, as might states, car makes and models, and so on. This kind of application-specific lexical clustering, whether done by hand or with the help of resources such as thesauri or semantic networks, should improve performance by overcoming inherent data sparseness problems. Classes might also prove helpful in dealing with changing items such as movie titles. In our earlier experiments, we used latent semantic analysis (Deerwester et al. 1990) for dimensionality reduction in an attempt to automatically cluster words that are semantically similar. This involved selecting dimensionality k, which is less than the rank r of the original term-document matrix. We found performance degrades for any $k < r$.

In the current version of our system, the interface between the automatic speech recognizer and the call router is the top hypothesis of the speech recognizer for the speech input. As reported in Table 3, this top hypothesis has an approximately 10% error rate on salient unigrams. One way to improve this error rate is to allow the speech recognizer to produce an n-best list of the top n recognition hypotheses or even a probabilistic word graph rather than a single best hypothesis. The n-gram terms can then be extracted from the graph in a straightforward manner and weighted according to their scores from the recognizer. Our prediction is that this will lead to increased recall, with perhaps a slight degradation in precision. However, since increased recall will, at the very least, increase the chance that the disambiguation module can formulate reasonable queries, we expect the system's overall performance to improve as a result.

7. Conclusions

We described and evaluated a domain-independent, automatically trained call router that takes one of three actions in response to a caller's request. It can route the call to a destination within the call center, attempt to dynamically formulate a disambiguation query, or route the call to a human operator. The routing module selects a set of candidate destinations based on n-gram terms extracted from the caller's request and a vector-based comparison between these n-gram terms and each possible destination. If disambiguation is necessary, a *yes-no* question or a *wh*-question is dynamically generated from among n-gram terms automatically extracted from the training data based on closeness, relevance, and disambiguating power. This query formulation process allows the system to tailor the disambiguating query to the caller's original request and the candidate destinations.

We have further demonstrated the effectiveness of our call router by evaluating the call router on both transcriptions of caller requests and the output of an automatic speech recognizer on these requests. When the input to the call router is free of recognition errors, our system correctly routes 93.8% of the calls after redirecting 10.2% of all calls to a human operator. When using the output of a speech recognizer with an approximately 23% word error rate, the rejection rate drops to 9.3%, the upper bound of the router performance drops from 97.2% to 92.5%, and the lower bound of the performance drops from 75.6% to 72.2%, illustrating the robustness of our call router in the face of speech recognition errors.

Acknowledgments

We would like to thank Christer Samuelsson, Chin-Hui Lee, Wu Chou, Ron Hartung, and Jim Hieronymus for helpful discussions, Wolfgang Reichl for providing us with speech recognition results, Jan van Santen for help with the statistics, as well as Christine Nakatani, Diane Litman, and the three anonymous reviewers for their helpful comments on an earlier draft of this paper.

References

Abella, Alicia and Allen L. Gorin. 1997. Generating semantically consistent inputs to a dialog manager. In *Proceedings of the 5th European Conference on Speech Communication and Technology*, pages 1,879–1,882.

Carletta, Jean C. 1996. Assessing the reliability of subjective codings. *Computational Linguistics*, 22(2):249–254.

Carpenter, Bob and Jennifer Chu-Carroll. 1998. Natural language call routing: A robust, self-organizing approach. In *Proceedings of the Fifth International Conference on Spoken Language Processing*, pages 2,059–2,062.

Chu-Carroll, Jennifer and Bob Carpenter. 1998. Dialogue management in vector-based call routing. In *Proceedings of the 36th Annual Meeting*, pages 256–262. Association for Computational Linguistics.

Deerwester, Scott, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407.

Gorin, Allen L. 1996. Processing of semantic information in fluently spoken language. In *Proceedings of the International Conference on Spoken Language Processing*, pages 1,001–1,004.

Gorin, Allen L., G. Riccardi, and J. H. Wright. 1997. How may I help you? *Speech Communication*, 23:113–127.

Green, Nancy and Sandra Carberry. 1994. A hybrid reasoning model for indirect answers. In *Proceedings of the 32nd Annual Meeting*, pages 58–65. Association for Computational Linguistics.

Hockey, Beth Ann, Deborah Rossen-Knill, Beverly Spejewski, Matthew Stone, and Stephen Isard. 1997. Can you predict responses to yes/no questions? Yes, no, and stuff. In *Proceedings of the 5th European Conference on Speech Communication and Technology*, pages 2,267–2,270.

Ittner, David J., David D. Lewis, and David D. Ahn. 1995. Text categorization of low quality images. In *Symposium on Document Analysis and Information Retrieval*, pages 301–315, Las Vegas.

Lewis, David D. and William A. Gale. 1994. A sequential algorithm for training text classifiers. In W. Bruce Croft and C. J. van Rijsbergen, editors, *ACM-SIGIR Conference on Research and Development in Information Retrieval*, pages 3–12, London. Springer-Verlag.

Reichl, Wolfgang, Bob Carpenter, Jennifer Chu-Carroll, and Wu Chou. 1998. Language modeling for content selection in human-computer dialogues. In *Proceedings of the Fifth International Conference on Spoken Language Processing*, pages 2,315–2,318.

Riccardi, G. and A. L. Gorin. 1998. Stochastic language models for speech recognition and understanding. In *Proceedings in the Fifth International Conference on Spoken Language Processing*, pages 2,087–2,090.

Salton, Gerald. 1971. *The SMART Retrieval System*. Prentice Hall, Inc.

Siegel, Sidney. and N. John Castellan, Jr. 1988. *Nonparametric Statistics for the Behavioral Sciences*. McGraw-Hill.

Sparck Jones, Karen. 1972. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28(1):11–20.

Sproat, Richard, editor. 1998. *Multilingual Text-to-Speech Synthesis: The Bell Labs Approach*. Kluwer, Boston, MA.

Walker, Marilyn A., Diane J. Litman, Candace A. Kamm, and Alicia Abella. 1998. Evaluating spoken dialogue agents with PARADISE: Two case studies. *Computer Speech and Language*, 12(3):317–347.

Wright, J. H., A. L. Gorin, and G. Riccardi 1997. Automatic acquisition of salient grammar fragments for call-type classification. In *Proceedings of the 5th European Conference on Speech Communication and Technology*, pages 1,419–1,422.