Dialogue Management in Vector-Based Call Routing

Jennifer Chu-Carroll and **Bob Carpenter**
Lucent Technologies Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974, U.S.A.
E-mail: {jencc,carp}@research.bell-labs.com

Abstract

This paper describes a domain independent, automatically trained call router which directs customer calls based on their response to an open-ended *"How may I direct your call?"* query. Routing behavior is trained from a corpus of transcribed and hand-routed calls and then carried out using vector-based information retrieval techniques. 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. Our approach is domain independent and the training process is fully automatic. Evaluations over a financial services call center handling hundreds of activities with dozens of destinations demonstrate a substantial improvement on existing systems by correctly routing 93.8% of the calls after punting 10.2% of the calls to a human operator.

1 Introduction

The *call routing* task involves directing a user's call to the appropriate destination within a call center or providing some simple information, such as loan rates. In current systems, the user's goals are typically gleaned via a touch-tone system employing a rigid hierarchical 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 options; these problems are compounded by the necessity of descending a nested hierarchy of choices to zero in on a particular activity. Even simple requests such as *"I'd like my savings account balance"* may require users to navigate as many as four or five nested menus with four or five options each. We have developed 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.

Human operators respond to a caller request by 1) routing the call to an appropriate destination, or 2) querying the caller for further information to determine where to route the call. Our automatic call router has these two options as well as a third option of sending the call to a human operator. The rest of this paper provides both a description and an evaluation of an automatic call router driven by vector-based information retrieval techniques. After introducing our fundamental routing technique, we focus on the disambiguation query generation module. Our disambiguation module is based on the same statistical training as routing, and dynamically generates queries tailored to the caller's request and the destinations with which it is consistent. The main advantages of our system are that 1) it is domain independent, 2) it is trained fully automatically to both route and disambiguate requests, and 3) its performance is sufficient for use in the field, substantially improving on that of previous systems.

2 Related Work

Call routing is similar to topic identification (McDonough et al., 1994) and document routing (Harman, 1995) in identifying which one of n topics (destinations) most closely matches a caller's request. Call routing is distinguished from these activities by requiring a single destination, but allowing a request to be refined in an interactive dialogue. We are further interested in carrying out the routing using natural, conversational language.

The only work on call routing to date that we are aware of is that by Gorin et al. (to appear). They select salient phrase fragments from caller requests, such as *made a long distance* and *the area code for*. These phrase fragments are used to determine the most likely destination(s), which they refer to as *call type(s)*, for the request either by computing the *a posteriori* probability for each call type or by passing the fragments through a neural network classifier. 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. Their intention is to utilize the outcome of this algorithm to select from a set of dialogue strategies for response generation.

3 Corpus Analysis

To examine human-human dialogue behavior in call routing, we analyzed a set of 4497 transcribed telephone calls involving customers interacting with human operators, looking at both the *semantics of caller requests* and

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

Table 1: Semantic Types of Caller Requests

	Notification	Query	
		NP	Others
# of calls	3608	657	232
% of all calls	80.2%	14.6%	5.2%

Table 2: Call Operator Dialogue Actions

Figure 1: Call Router Architecture

dialogue actions for response generation. The call center provides financial services in hundreds of categories in the general areas of banking, credit cards, loans, insurance and investments; we concentrated on the 23 destinations for which we had at least 10 calls in the corpus.

3.1 Semantics of Caller Requests

The operator provides an open-ended prompt of *"How may I direct your call?"* We classified user responses into three categories. First, callers may explicitly provide a **destination name**, either by itself or embedded in a complete sentence, such as *"may I have consumer lending?"* Second, callers may describe the **activity** they would like to perform. Such requests may be unambiguous, such as *"I'd like my checking account balance"*, or ambiguous, such as *"car loans please"*, which in our call center can be resolved to either *consumer lending*, which handles new car loans, or to *loan services*, which handles existing car loans. Third, a caller can provide an **indirect request**, in which they describe their goal in a roundabout way, often including irrelevant information. This often occurs when the caller either is unfamiliar with the call center hierarchy or does not have a concrete idea of how to achieve the goal, as in *"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 destination name or activity. Since there is a fairly small number (dozens to hundreds) of activities being handled by each destination, requests based on name and activity are expected to be more predictable and thus more suitable for handling by an automatic call router. Thus, our goal is to automatically route those calls based on name and activity, while leaving the indirect or inappropriate requests to human call operators.

3.2 Dialogue Actions for Response Generation

We also analyzed the operator's responses to caller requests to determine the dialogue actions needed for response generation in our automatic call router. We found that in the call routing task, the call operator either *notifies* the customer of the routing destination or asks a disambiguating *query*.[1]

---

[1] In cases where the operator generates an acknowledgment, such as *uh-huh*, midway through the caller's request, we analyzed the next operator utterance.
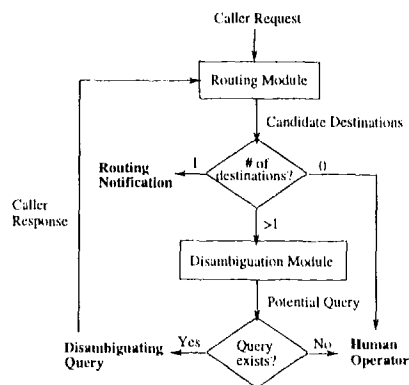
Table 2 shows the frequency that each dialogue action should be employed based strictly on the presence of ambiguity in the caller requests in our corpus. We further analyzed those calls considered ambiguous within our call center and noted that 75% of such ambiguous requests involve underspecified noun phrases, such as requesting *car loans* without specifying whether it is an *existing* or *new* car loan. The remaining 25% of the ambiguous requests involve underspecified verb phrases, such as asking to *transfer funds* without specifying the types of accounts to and from which the transfer will occur, or missing verb phrases, such as asking for *direct deposit* without specifying whether the caller wants to *set up* or *change an existing* direct deposit.

4 Dialogue Management in Call Routing

Our call router consists of two components: the *routing* module and the *disambiguation* module. The routing module takes a caller request and determines a set of destinations to which the call can reasonably be routed. If there is exactly one such destination, the call is routed there and the customer notified; if there are multiple destinations, the disambiguation module is invoked in an attempt to formulate a query; and if there is no appropriate destination or if a reasonable disambiguation query cannot be generated, the call is routed to an operator. Figure 1 shows a diagram outlining this process.

4.1 The Routing Module

Our approach is novel in its application of information retrieval techniques to select candidate destinations for a call. We treat call routing as an instance of *document routing*, where a collection of judged documents is used for training and the task is to judge the relevance of a set of test documents (Schütze et al., 1995). More specifi-

257

cally, each destination in our call center is represented as a collection of documents (transcriptions of calls routed to that destination), and given a caller request, we judge the relevance of the request to each destination.

4.1.1 The Training Process

Document Construction Our training corpus consists of 3753 calls each of which is hand-routed to one of 23 destinations.[2] Our first step is to create one (virtual) document per destination, which contains the text of the callers' contributions to all calls routed to that destination.

Morphological Filtering We filter each (virtual) document through the morphological processor of the Bell Labs' Text-to-Speech synthesizer (Sproat, 1997) to extract the root form of each word in the corpus. Next, the root forms of caller utterances are filtered through two lists, the *ignore list* and the *stop list*, in order to build a better n-gram model. The ignore list consists of noise words, such as *uh* and *um*, which sometimes get in the way of proper n-gram extraction, as in *"I'd like to speak to someone about a car uh loan"*. With noise word filtering, we can properly extract the bigram *"car,loan"*. The stop list enumerates words that do not discriminate between destinations, such as *the*, *be*, and *afternoon*. 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 the training corpus. Note that when a stop word is filtered out of the caller utterance, a placeholder is inserted to prevent the words preceding and following the stop word to form n-grams. For instance, after filtering the stop words out of *"I want to check on an account"*, the utterance becomes *"<sw> <sw> <sw> check <sw> <sw> account"*. Without the placeholders, we would extract the bigram *"check,account"*, just as if the caller had used the term *checking account*.

Term Extraction We extract the n-gram terms that occur more frequently than a pre-determined threshold and do not contain any stop words. Our current system uses unigrams that occurred at least twice and bigrams and trigrams that occurred at least three times in the corpus. No 4-grams occurred three times.

Term-Document Matrix Once the set of relevant terms is determined, we construct an $m \times n$ term-document frequency matrix A whose rows represent the m terms, whose columns represent the n destinations, and where an entry $A_{t,d}$ is the frequency with which term t occurs in calls to destination d.

It is often advantageous to weight the raw counts to fine tune the contribution of each term to routing. We begin by normalizing the row vectors representing terms by making them each of unit length. Thus we divide each row A_t in the original matrix by its length,

---

[2]These 3753 calls are a subset of the corpus of 4497 calls used in our corpus analysis. We excluded those ambiguous calls that were not resolved by the operator.

$(\sum_{1 \le e \le n} A_{t,e}^2)^{1/2}$. Our second weighting is based on the notion that a term that only occurs in a few documents is more important in discriminating among documents than a term that occurs in nearly every document. We use 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, by means of $IDF(t) = \log_2 n/d(t)$ where t is a term, n is the total number of documents in the corpus, and $d(t)$ is the number of documents containing the term t. Thus we obtain a weighted matrix B, whose elements are given by $B_{t,d} = A_{t,d} \times IDF(t)/(\sum_{1 \le e \le n} A_{t,e}^2)^{1/2}$.

Vector Representation To reduce the dimensionality of our vector representations for terms and documents, we applied the singular value decomposition (Deerwester et al., 1990) to the $m \times n$ matrix B of weighted term-document frequencies. Specifically, we take $B = USV^{\mathrm{T}}$, where U is an $m \times r$ orthonormal matrix (where r is the rank of B), V is an $n \times r$ orthonormal matrix, and S is an $r \times r$ diagonal matrix such that $s_{1,1} \ge s_{2,2} \ge \cdots \ge s_{r,r} > 0$.

We can think of each row in U as an r-dimensional vector that represents a term, whereas each row in V is an r-dimensional vector representing a document. With appropriate scaling of the axes by the singular values on the diagonal of S, we can compare documents to documents and terms to terms using their corresponding points in this new r-dimensional space (Deerwester et al., 1990). For instance, to employ the dot product of two vectors as a measure of their similarity as is common in information retrieval (Salton, 1971), we have the matrix $B^{\mathrm{T}}B$ whose elements contain the dot product of document vectors. Because S is diagonal and U is orthonormal, $B^{\mathrm{T}}B = VS^2V^{\mathrm{T}} = VS(VS)^{\mathrm{T}}$. Thus, element i, j in $B^{\mathrm{T}}B$, representing the dot product between document vectors i and j, can be computed by taking the dot product between the i and j rows of the matrix VS. In other words, we can consider rows in the matrix VS as vectors representing documents for the purpose of document/document comparison. An element of the original matrix $B_{i,j}$, representing the degree of association between the ith term and the jth document, can be recovered by multiplying the ith term vector by the jth scaled document vector, namely $B_{i,j} = U_i((VS)_j)^{\mathrm{T}}$.

4.1.2 Call Routing

Given the vector representations of terms and documents (destinations) in r-dimensional space, how do we determine to which destination a new call should be routed? Our process for vector-based call routing consists of the following four steps:

Term Extraction Given a transcription of the caller's utterance (either from a keyboard interface or from the output of a speech recognizer), the first step is to extract the relevant n-gram terms from the utterance. For instance, term extraction on the request *"I want to check the balance in my savings account"* would result in

one bigram term, *"saving,account"*, and two unigrams, *"check"* and *"balance"*.

Pseudo-Document Generation Given the extracted terms from a caller request, we can represent the request as an m-dimensional vector Q where each component Q_i represents the number of times that the ith term occurred in the caller's request. We then create an r-dimensional *pseudo-document* vector $D = QU$, following the standard methodology of vector-based information retrieval (see (Deerwester et al., 1990)). Note that D is simply the sum of the term vectors U_i for all terms occurring in the caller's request, weighted by their frequency of occurrence in the request, and is scaled properly for document/document comparison.

Scoring Once the vector D for the pseudo-document is determined, we compare it with the document vectors by computing the cosine between D and each scaled document vectors in VS. Next, we transform the cosine score for each destination using a sigmoid function specifically fitted for that destination to obtain a confidence score that represents the router's confidence that the call should be routed to that destination.

The reason for the mapping from cosine scores to confidence scores is because the absolute degree of similarity between a request and a destination, as given by the cosine value between their vector representations, 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. Thus we collected, for each destination, a set of cosine value/routing value pairs over all calls in the training data, where the routing value is 1 if the call should be routed to that destination and 0 otherwise. Then for each destination, we used the least squared error method in fitting a sigmoid function, $1/(1 + e^{-(ax+b)})$, to the set of cosine/routing pairs.

We tested the routing performance using cosine vs. confidence values on 307 unseen unambiguous requests. In each case, we selected the destination with the highest cosine/confidence score to be the target destination. Using strict 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 a relative reduction in error rate of 16.7%.

Decision Making The outcome of the routing module is a set of destinations whose confidence scores are above a pre-determined threshold. These candidate destinations represent those to which the caller's request can reasonably be routed. If there is only one such destination, then the call is routed and the caller notified; if there are two or more possible destinations, the disambiguation module is invoked in an attempt to formulate a query; otherwise, the the call is routed to an operator.

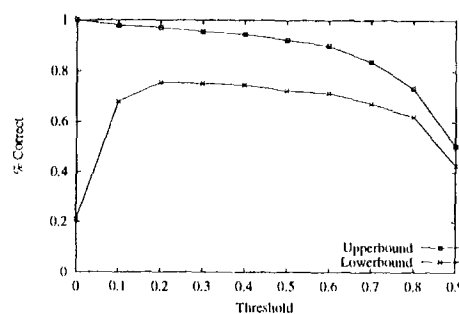To determine the optimal value for the threshold, we

Figure 2: Router Performance vs. Threshold

ran a series of experiments to compute the upperbound and lowerbound of the router's performance varying the threshold from 0 to 0.9 at 0.1 intervals. The lowerbound represents the percentage of calls that are routed correctly, while the upperbound indicates the percentage of calls that have the potential to be routed correctly after disambiguation (see section 5 for details on upperbound and lowerbound measures). The results in Figure 2 show 0.2 to be the threshold that yields optimal performance.

4.2 The Disambiguation Module

The disambiguation module attempts to formulate an appropriate query to solicit further information from the caller in order to determine a unique destination to which the call should be routed. To generate an appropriate query, the caller's request and the candidate destinations must both be taken into account. We have developed a vector-based method for dynamically generating disambiguation queries by first selecting a set of terms and then forming a *wh* or *yes-no* question from these selected terms.

The terms selected by the disambiguation mechanism are those terms related to the original request that can likely be used to disambiguate among the candidate destinations. These terms are chosen by filtering all terms based on the following three criteria:

1. **Closeness**: We choose terms that are close (by the cosine measure) to the differences between the scaled pseudo-document query vector, D, and vectors representing the candidate destinations in VS. The intuition is that adding terms close to the differences will disambiguate the original query.

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

3. **Disambiguating power**: Finally, we restrict attention to relevant terms that can be added to the

259

original request to result in an unambiguous routing using the routing mechanism described in Section 4.1.2. If none of the relevant terms satisfy this criterion, then we include all relevant terms in the set of disambiguating terms. Thus, instead of giving up the disambiguation process when no one term is predicted to resolve the ambiguity, the system may pose a question which further specifies the request and then select a disambiguating term based on this refined (although still ambiguous) request.

The result of this filtering process is a finite set of terms which are relevant to the original ambiguous query and, when added to it, are likely to resolve the ambiguity. If a significant number of these terms share a head word, such as *loan*, the system asks the wh-question *"for what type of loan?"* Otherwise, the term that occurred most frequently in the training data is selected, based on the heuristic that a more common term is likely to be relevant than an obscure term, and a yes-no question is formed based on this term. A third alternative would be to ask a disjunctive question, but we have not yet explored this possibility. Figure 1 shows that after the system poses its query, it attempts to route the refined request, which is the original request augmented with the caller response to the system's query. In the case of wh-questions, n-gram terms are extracted from the caller's response. In the case of yes-no questions, the system determines whether a *yes* or *no* answer is given.[3] In the former case, the disambiguating term used to form the query is considered the caller response, while in the latter case, the response is treated as in responses to wh-questions.

Note that our disambiguation mechanism, like our basic routing technique, is fully domain-independent. It utilizes a set of n-gram terms, as well as term and document vectors that were obtained by the training of the call router. Thus, porting the call router to a new domain requires no change in the disambiguation module.

4.3 Example

To illustrate our call router, consider the request *"loans please."* This request is ambiguous because our call center handles *mortgage loans* separately from all other types of loans, and for all other loans, *existing loans* and *new loans* are again handled by different departments.

Given this request, the call router first extracts the relevant n-gram terms, which in this case results in the unigram *"loan"*. It then computes a pseudo-document vector that represents this request, which is compared in turn with the 23 vectors representing all destinations in the call center. The cosine values between the request and each destination are then mapped into confidence values.

Using a confidence threshold of 0.2, we have two candidate destinations, *Loan Servicing* 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 between the request vector and either of the two candidate destination vectors. 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. 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 disambiguation power, resulting in 18 disambiguating terms. Since 11 of these disambiguating terms share a head noun *loan*, a wh-question is generated based on this head word, 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"* to the original request and attempts to route the refined request. This refined request is again ambiguous between *Loan Servicing* and *Consumer Lending* since 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 term *"exist,car,loan"*. Thus, the system generates the yes-no question *"is this about an existing car loan?"*[4] If the user responds *"yes"*, then the trigram term *"exist,car,loan"* is added to the refined request and the call routed to *Loan Servicing*; if the user says *"no, it's a new car loan"*, then *"new,car,loan"* is extracted from the response and the call routed to *Consumer Lending*.

5 Evaluation

5.1 The Routing Module

We performed an evaluation of the routing module of our call router on a fresh set of 389 calls to a human operator.[5] Out of the 389 requests, 307 are unambiguous and routed to their correct destinations, and 82 were ambiguous and annotated with a list of candidate destinations. Unfortunately, in this test set, only the caller's first utterance was transcribed. Thus we have no information about where the ambiguous calls were eventually routed.

The routing decision made for each call is classified into one of 8 groups, as shown in Figure 3. For instance,

---

[3]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; 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.

[4]Our current system uses simple template filling for response generation by utilizing a manually constructed mappings from n-gram terms to their inflected forms, such as from *"exist,car,loan"* to *"an existing car loan"*.

[5]The calls in the test set were recorded separately from our training corpus. In this paper, we focus on evaluation based on transcriptions of the calls. A companion paper compares call performance on transcriptions to the output of a speech recognizer (Carpenter and Chu-Carroll, submitted).
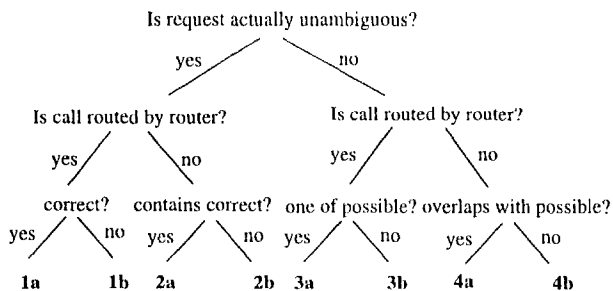
Figure 3: Classification of Router Outcome

	Unambiguous Requests	Ambiguous Requests	All Requests
LB	1a/(1+2)	4a/(3+4)	(1a+4a)/all
UB	(1a+2a)/(1+2)	(3a+4a)/(3+4)	(1a+2a+3a+4a)/all

Table 3: Calculation of Upperbounds and Lowerbounds

group **1a** contains those calls which are 1) actually unambiguous, 2) considered unambiguous by the router, and 3) routed to the correct destination. On the other hand, group **3b** contains those calls which are 1) actually ambiguous, 2) considered by the router to be unambiguous, and 3) routed to a destination which 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 lowerbound performance, which measures the percentage of calls that are correctly routed, and an upperbound performance, which measures the percentage of calls that are either correctly routed or have the potential to be correctly routed. Table 3 shows how the upperbounds and lowerbounds are computed based on the classification in Figure 3 for each of the three data sets. For instance, for unambiguous requests (classes 1 and 2), the lowerbound is the number of calls actually routed to the correct destination (1a) divided by the number of total unambiguous requests, while the upperbound is the number of calls actually routed to the correct destination (1a) plus the number of calls which the router finds to be ambiguous between the correct destination and some other destination(s) (2a), divided by the number of unambiguous queries. The calls in category 2a are considered to be potentially correct because it is likely that the call will be routed to the correct destination after disambiguation.

Table 4 shows the upperbound and lowerbound performance for each of the three test sets. These results show

	Unambiguous Requests	Ambiguous Requests	All Requests
LB	80.1%	58.5%	75.6%
UB	96.7%	98.8%	97.2%

Table 4: Router Performance with Threshold = 0.2

that the system's overall performance will fall somewhere between 75.6% and 97.2%. 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 16.6% of the unambiguous calls that were considered ambiguous by the router (class 2a), and 2) the percentage of calls that were routed correctly out of the 40.4% ambiguous calls that were considered unambiguous and routed by the router (class 3a). Note that the performance figures in Table 4 are the result of 100% automatic routing, since no request in our test set failed to evoke at least one candidate destination. In the next sections, we discuss the performance of the disambiguation module, which determines the overall system performance, and show how allowing calls to be punted to operators affects the system's performance.

5.2 The Disambiguation Module

To evaluate our disambiguation module, we needed dialogues which satisfy two criteria: 1) the caller's first utterance is ambiguous, and 2) the operator asked a follow-up question to disambiguate the query and subsequently routed the call to the appropriate destination. We used 157 calls that meet these two criteria as our test set. Note that this test set is disjoint from the test set used in the evaluation of the router (Section 5.1), since none of the transcribed calls in the latter test set satisfied criterion (2).

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

1. Unambiguous: in this case 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.

2. Ambiguous: in this case the router attempted to initiate disambiguation. The outcome of the routing of these calls were determined as follows:

   (a) Correct, if a disambiguation query was generated which, when answered, led to the correct destination.

   (b) Incorrect, if a disambiguation query was generated which, when answered, could not lead to a correct destination.

   (c) 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 an operator.

Table 5 shows the number of calls that fall into each of the 5 categories. Out of the 157 calls, the router automatically routed 115 of them either with or without disambiguation (73.2%). Furthermore, 87.0% of these routed calls were routed to the correct destination. Notice that out of the 52 ambiguous calls that the router considered unambiguous, 40 were routed correctly (76.9%).

261

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

Table 5: Performance of Disambiguation Module on Ambiguous Calls

	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%

Table 6: Overall Performance of Call Router

This is simply because our vector-based router is able to distinguish between cases where an ambiguous query is equally likely to be routed to more than one destination, 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 callers without disambiguation.

5.3 Overall Performance

Combining results from Section 5.2 for ambiguous calls with results from Section 5.1 for unambiguous calls leads to the overall performance of the call router in Table 6. The table shows the number of calls that will be correctly routed, incorrectly routed, and rejected, if we apply the performance of the disambiguation module (Table 5) to the calls that fall into each class in the evaluation of the routing module (Section 5.1). Our results show that out of the 389 calls in our test set, 89.8% of the calls will be automatically routed by the call router. Of these calls, 93.8% (which constitutes 84.2% of all calls) will be routed to their correct destinations. This is substantially better than the results obtained by Gorin et al., who report an 84% correct routing rate with a 10% false rejection rate (routed to an operator when the call could have been automatically routed) on 14 destinations (Gorin et al., to appear).[6]

6 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

[6]Gorin et al.'s results are measured without the possibility of system queries. To provide a fair comparison, we evaluated our routing module on all 389 calls in our test set using the scoring method described in (Gorin et al., to appear) (which corresponds roughly to our upperbound measure), and achieved a 94.1% correct routing rate to 23 destinations when all calls are automatically routed (no false rejection), a substantial improvement over their system.

formulate a disambiguating query, or route the call to a human operator. The routing module of the call router selects a set of candidate destinations based on n-gram terms extracted from the caller request and a vector-based comparison between these n-gram terms and each possible destination. If disambiguation is necessary, a yes-no question or wh-question is dynamically generated from among known n-gram terms in the domain based on closeness, relevance, and disambiguating power, thus tailoring the disambiguating query to the original request and the candidate destinations. Finally, our system performs substantially better than the best previously existing system, achieving an overall 93.8% correct routing rate for automatically routed calls when rejecting 10.2% of all calls.

References

A. Abella and A. Gorin. 1997. Generating semantically consistent inputs to a dialog manager. In *Proc. EUROSPEECH*, pages 1879–1882.

B. Carpenter and J. Chu-Carroll. submitted. Natural language call routing: A robust, self-organizing approach.

S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41:391–407.

A. Gorin, G. Riccardi, and J. Wright. to appear. How may I help you? *Speech Communication*.

N. Green and S. Carberry. 1994. A hybrid reasoning model for indirect answers. In *Proc. ACL*, pages 58–65.

D. Harman. 1995. Overview of the fourth Text REtrieval Conference. In *Proc. TREC*.

B. Hockey, D. Rossen-Knill, B. Spejewski, M. Stone, and S. Isard. 1997. Can you predict responses to yes/no questions? yes, no, and stuff. In *Proc. EUROSPEECH*, pages 2267–2270.

J. McDonough, K. Ng, P. Jeanrenaud, H. Gish, and J. R. Rohlicek. 1994. Approaches to topic identification on the switchboard corpus. In *Proc. ICASSP*, pages 385–388.

G. Salton. 1971. *The SMART Retrieval System*. Prentice Hall.

H. Schütze, D. Hull, and J. Pedersen. 1995. A comparison of classifiers and document representations for the routing problem. In *Proc. SIGIR*.

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

R. Sproat, editor. 1997. *Multilingual Text-to-Speech Synthesis: The Bell Labs Approach*. Kluwer.