User Model User-Adap Inter (2007) 17:93–117
DOI 10.1007/s11257-006-9021-6

ORIGINAL PAPER

Improving command and control speech recognition on mobile devices: using predictive user models for language modeling

Tim Paek · David Maxwell Chickering

Received: 1 November 2005 / Accepted in revised form: 11 July 2006 /
Published online: 18 January 2007

Abstract Command and control (C&C) speech recognition allows users to interact with a system by speaking commands or asking questions restricted to a fixed grammar containing pre-defined phrases. Whereas C&C interaction has been commonplace in telephony and accessibility systems for many years, only recently have mobile devices had the memory and processing capacity to support client-side speech recognition. Given the personal nature of mobile devices, statistical models that can predict commands based in part on past user behavior hold promise for improving C&C recognition accuracy. For example, if a user calls a spouse at the end of every workday, the language model could be adapted to weight the spouse more than other contacts during that time. In this paper, we describe and assess statistical models learned from a large population of users for predicting the next user command of a commercial C&C application. We explain how these models were used for language modeling, and evaluate their performance in terms of task completion. The best performing model achieved a 26% relative reduction in error rate compared to the base system. Finally, we investigate the effects of personalization on performance at different learning rates via online updating of model parameters based on individual user data. Personalization significantly increased relative reduction in error rate by an additional 5%.

Keywords Command and control · Language modeling · Speech recognition · Predictive user models

T. Paek (✉) · D. M. Chickering
Microsoft Research, Redmond, WA, USA
e-mail: timpaek@microsoft.com

D. M. Chickering
e-mail: dmax@microsoft.com

1 Introduction

Since the beginning of research in spoken language understanding in the 1970s (Woods 1985), people have dreamed of communicating with all kinds of devices and appliances using speech recognition. Of the many types of interaction that speech recognition affords, command and control (C&C) is perhaps the simplest, both in terms of the interaction model and the required technology. With regard to the interaction model, C&C allows users to interact with a system by simply speaking commands or asking questions restricted to a fixed grammar containing pre-defined phrases. With regard to the required technology, all that is essential for basic operation, other than a speech engine, is a context-free grammar (CFG), which is a formal specification of rules allowing for embedded recursion that defines the set of allowable utterances (Manning and Schütze 1999). Although statistical language models are prevalent in research, most commercial applications employ either a CFG or probabilistic CFG (PCFG)—a CFG with probabilities added to the rules, as standardized in the Speech Recognition Grammar Specification (SRGS) format (W3C 2004).

With the introduction of voice dialing on mobile devices, C&C interaction successfully reached the wider consumer market. While C&C interaction has been commonplace in telephony and accessibility systems for many years, only recently have mobile devices had the memory and processing capacity to support not only speech recognition, but a whole range of multimedia functionalities that could be controlled by speech.

The mobile platform is well suited for exploiting the advantages of speech over other input modalities because speech: (1) facilitates easier access to information through description rather than deictic reference, and (2) acts as an "ambient medium" (Rosenfeld et al. 2001) that allows users to do other things at the same time, instead of requiring more focused attention. These advantages help explain why voice dialing on mobile devices is so compelling. Perusing a large database of contacts and selecting an item of interest is not only inherently more difficult than simply describing the item you want (which people often know by heart), but it is orders of magnitude worse with confined input or limited screen real estate. Add to that the typical user scenario of divided attention and multitasking, where hands and eyes (i.e., referential devices) may be busy, and speech often prevails as the preferred modality (see Jameson and Klöckner 2004 for a discussion of where speech is not preferred).

Speech has yet another advantage on mobile devices: it can be scaled down to much smaller and cheaper form factors than can visual or manual modalities (Rosenfeld et al. 2001). Because the trend in mobile devices is to develop smaller hardware with greater functionality, it behooves researchers to explore how best to improve C&C speech recognition on mobile devices. According to market research on cell phones, consumers are more interested in using voice input as a feature than on any other input enhancements such as touch screens (Strother 2005). Unfortunately, when speech recognition errors occur, user frustration runs high and many people give up on voice input entirely.

One distinctive aspect of mobile devices that could serve as both a means to improve C&C interaction as well as to create an affective connection with users is personalization. Mobile devices are typically used by just the owner. Pride of ownership is evident in the way users personalize their devices through backgrounds, skins, ring tones, etc. People generally tend to be creatures of habit, so it is possible that individual users may display systematic patterns of usage for their personal devices.

This kind of information could be modeled with machine learning techniques and leveraged to improve the performance of C&C speech recognition. For example, if a user calls a spouse at the end of every workday, the language model could be adapted to weight that spouse more than other contacts during that time.

If enough people exhibit similar patterns of usage for their devices, leveraging models of the general population may be just as effective for improving recognition performance. As researchers in the user modeling community have shown, whether or not individual user models can outperform general user models is an empirical question (Jameson and Wittig 2001). In this paper, we explore these kinds of questions and intuitions in depth.

This paper divides into three parts. In the first part, we describe and assess statistical models learned from a large population of users for predicting the next user command of a commercial C&C application. In the second part, we explain how these models were used for language modeling, and evaluate their performance in terms of task completion. Finally, in the third part, we investigate the effects of personalization on performance via online updating of model parameters based on individual user data.

2 Related research

In the field of speech understanding, most of the research in language modeling focuses on statistical language models such as n-grams (Jelinek 1995). In order to improve recognition accuracy for a particular user with statistical language models, these models need to be trained, at least in part, on user generated text. Unfortunately, most commercial applications utilize a CFG where what can be said has been deterministically fixed, though ideally, the grammar has been authored so that its coverage is sufficiently broad. Researchers interested in adapting a commercial application usually do so with a PCFG. For example, in a speech-enabled email application, Yu et al. (2003) have shown that adapting a PCFG according to individual email usage behavior could dramatically reduce error rate for proper name recognition, a task which is often more difficult than word recognition due to highly confusable and difficult to pronounce names. In particular, they estimated the probability of emailing a contact using just the frequencies of previous emails, augmented with smoothing and an exponential window that allows forgetting.

The work that we present can be viewed as extending their research in two ways. First, the statistical models we built for predicting contacts captured more than just the frequencies within a time frame. We generated as many features as possible and used machine learning algorithms to select the most salient ones for predicting the next contact on a mobile device. Second, while Yu et al. (2003) conducted experiments on a recognition task with over 100,000 contacts, we were able to find improvements on a C&C task with a much smaller domain, where users had anywhere between 0 to about 2,000 contacts.

In the field of user modeling, Horvitz and Shwe (1995) modeled the likelihood of different commands in a C&C decision support system as a function of GUI display context and user experience, though they did not formally evaluate whether there was any improvement in performance. User models for representing uncertainty about goals, beliefs and knowledge states have also played an important role in guiding several dialogue systems (Johansson 2002). With respect to C&C interaction in particular, Paek and Horvitz (2000), Horvitz and Paek (2001), and Chickering and

Paek (2005) modeled user intentions for various domains using Bayesian networks and influence diagrams that incorporated many features from the speech recognizer. In their work, user models were directly integrated into influence diagrams for decision making. In applying user models to language modeling, we leave the decision making about clarification dialogue in the hands of application developers, though our results in Sect. 5 assume the execution of the most likely command instead of a dialog repair. As such, our research is complementary to their line of work on dialogue management.

3 User model preliminaries

In this section, we describe how we modeled the domain of the commercial C&C speech application. Before delving into the details, however, we present an intuitive example of how user modeling can be exploited to improve speech recognition.

3.1 Example

In Sect. 1, we considered a user who calls a spouse at the end of every workday. Suppose that the spouse is named "Ellen" and that the user also has a colleague named "Allen". Suppose that when the user utters the command, "Call Ellen", the speech recognizer assigns probabilities of 0.9 to "Allen" and 0.8 to "Ellen". Because speech recognizers are typically speaker-independent, this kind of error can occur if the user exhibits atypical pronunciation due to, for instance, an accent. Without a user model, every time the user tries to call his spouse, he will experience the frustration of having the system confirm whether he wants to call his colleague.

Now, suppose that a statistical user model keeps track of who the user calls and when. If the user calls his spouse more frequently than his colleague, then when the user utters, "Call Ellen", even if the speech recognizer assigns the usual probabilities, we can moderate those probabilities by those obtained from the statistical user model. For example, suppose that the user model assigns 0.4 to Allen and 0.8 to Ellen. In that case, the system will take "Ellen" as the preferred interpretation because $0.8 \cdot 0.8 > 0.9 \cdot 0.4$. What is important is that the user model gets the relative probabilities correct. If it assigns 0.7 to both Allen and Ellen, then the system would perform exactly the same as before.

The value of exploiting a user model can also be seen in this example if the model has been trained to consider other factors, such as whether or not the call is being made on a weekend. If so, then the system can learn to take "Allen" as the preferred interpretation on weekdays and "Ellen" on weekends for the same token utterance.

3.2 Modeling the domain

The domain of our user modeling encompasses the functionality of a commercial C&C speech application for Pocket PC and Smart Phone devices called Microsoft *Voice Command*. With Voice Command, users can look up contacts, place phone calls, get calendar information, get device status information, control media and launch applications. Voice Command currently utilizes a speaker-independent, embedded speech engine; that is, it is geared for users of all types of voices and does not adapt its acoustic model to any particular user. As such, speech recognition errors pose a serious threat to user satisfaction and customer loyalty.

Because statistical models define probability distributions over random variables, our first step is to define the random variables corresponding to our domain. To simplify presentation, we adopt the following syntactic conventions. We denote a random variable by an upper-case token (e.g., A, Π) and a state or value of that variable by the same token in lower case (e.g., a, π). We denote sets of variables with bold-face capitalized tokens (e.g., $\mathbf{A}, \boldsymbol{\Pi}$) and corresponding sets of values by bold-face lower case tokens (e.g., $\mathbf{a}, \boldsymbol{\pi}$).

Importantly, our statistical user models define probability distributions over the *intent* or *goal* of the next user command in the application. For example, one goal might be to check the date. The application allows the user to express this goal in a number of ways, such as "*What is the current date?*" and "*What day is it today?*" Our models do not distinguish between the wordings of the same goal.

For modeling purposes, we find it convenient to decompose each user goal into two distinct parts. The first part of the user goal is defined by one of 17 different *predicate functions* (or *predicates* for short) that the application supports at the start of a C&C interaction. Table 1 lists the predicate functions by category along with example commands. The second part of the user goal is the (possibly constant) predicate *argument*. Some predicates, such as *Call*, take arguments based on content present on the device, such as a contact list. For all other predicates, the argument is constant and the predicate itself defines a goal.

To clarify our decomposition of user goals, consider the tree shown in Fig. 1. The children of the root node (circles and squares) correspond to predicates, and the children of each predicate (diamonds) correspond to possible argument values. In Voice Command, the argument values consist of start menu items for the *Launch* predicate, contact items for the *Call* and *Show* predicates, and media items for the *Play* predicate. For leaf-node predicates (squares), the argument is constant, such as the *Battery* predicate. Note that the set of all user goals corresponds to the set of leaves in the tree.

More formally, let Π denote the random variable whose values are the 17 different predicates available at the start of an interaction. For any predicate π, let A_π denote

Table 1 List of predicates, organized by category, supported in the C&C application along with example commands

Predicate	Category	Example
1. Launch	APPLICATION	"Start Pocket Word."
2. Reminders	CALENDAR	"Reminders off."
3. Appointment	CALENDAR	"What's my next meeting?"
4. General	MEDIA	"What song is this?" "Next track."
5. Play	MEDIA	"Play jazz." "Play Missa Solemnis."
6. Shuffle	MEDIA	"Turn shuffle on."
7. Call	PHONE	"Call Max." "Redial"
8. Dial	PHONE	"Dial four two five..."
9. Show	PHONE	"Show Max Chickering."
10. Battery	STATUS	"What's the battery level?"
11. Date	STATUS	"What is the current date?"
12. Flight Mode	STATUS	"Set flight mode off."
13. Missed Calls	STATUS	"What calls did I miss?"
14. Ringer	STATUS	"Set ringer to soft."
15. Set Profile	STATUS	"Set profile to Max."
16. Signal	STATUS	"What's my signal strength?"
17. Time	STATUS	"What time is it?"

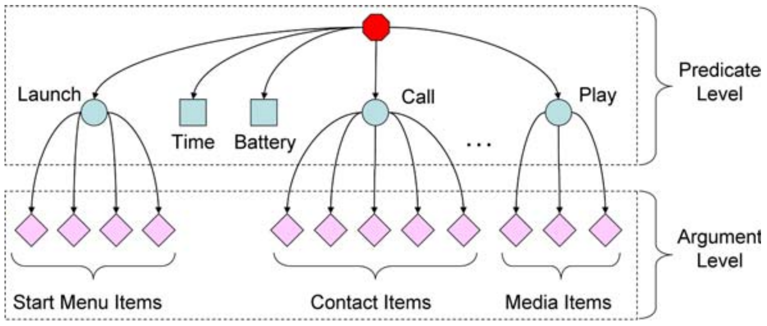

Fig. 1 A decomposition of user goals into a tree where the children of the root constitute predicates and their children constitute arguments

the random variable whose values are the possible argument values for predicate π; for a predicate π with a constant argument, we define A_π to be the constant $\emptyset$. For example, if π is the *Launch* predicate, then A_π is a variable whose values correspond to the set of start menu items installed on the mobile device that can be launched using C&C. If π is the *Time* predicate, then A_π is a constant equal to $\emptyset$.

Let G denote the random variable corresponding to the next user goal. We use $\langle$*predicate*, *argument*$\rangle$ pairs to denote the values of G. For example, to denote the user goal, $g_{\langle Launch,Excel\rangle}$, of launching Excel, we have

$$g_{\langle Launch,Excel\rangle} = \{\Pi = \pi_{Launch}, A_{Launch} = Excel\}$$

Similarly, to denote the user goal, $g_{\langle Time\rangle}$ of checking the time, we have

$$g_{\langle Time\rangle} = \{\Pi = \pi_{Time}, A_{Time} = \emptyset\}$$

As previously described, all user goals correspond to leaves in the tree shown in Fig. 1. By denoting each value g as a pair, we are simply describing each leaf by the corresponding path from the root node to the leaf.

For any value g of the random variable G, we use $\pi(g)$ and $a(g)$ to denote the individual predicate and argument value, respectively, of the pair.

Our model uses a number of feature variables to predict the next user goal G. Sect. 3.4.1 describes all the features in detail. For modeling purposes, we use $\boldsymbol{F}$ to denote the set of feature variables that we are able to observe when we predict G. Conditioning the next user goal on observed feature variables, our model defines the posterior probability over G as follows:

$$\begin{aligned} &p(G = g|\boldsymbol{F} = \boldsymbol{f}) \\ &\quad = p(\Pi = \pi(g)|\boldsymbol{F} = \boldsymbol{f}) \times p(A_{\pi(g)} = a(g)|\Pi = \pi(g), \boldsymbol{F} = \boldsymbol{f}) \end{aligned} \tag{1}$$

We call the first term in the right-hand side of Eq. 1 the *predicate model*, and the second term in the right-hand side of Eq. 1 the *argument model*. Note that the argument model is a function of the predicate. Also note that when the argument variable $A_{\pi(g)}$ for a goal g is constant, the argument model is simply defined to be unity.

3.3 Data collection

In order to learn various types of predicate and argument models for Eq. 1, we collected over 1.2 GB of data from 204 users of the C&C application. The users' occupational backgrounds spanned the general population of Pocket PC and Smart Phone owners. Each user contributed anywhere from 5 to over 200 commands, saved out as sound (.wav) files. Each sound file had a time stamp, which was critical for picking out systematic patterns of behavior. We also logged whatever personal data their devices contained at the time of each command (i.e., start menu items, contact items, and media items), as well as user preferences for what types of commands they had enabled and disabled in the control panel settings.

All sound files were transcribed by a paid professional transcription service. Transcriptions not only included the actual spoken text, if any, but also disfluencies as well as different types of noise, such as people speaking in the background or coughing. We ignored all transcriptions that did not have an associated command; the majority of such cases came from accidental pressing of the push-to-talk button.

3.4 Model selection

For model selection, we chose to learn decision trees, where the target variable was either the predicate or argument, as we elaborate in depth in Sect. 4. Our choice was partly motivated by the need to make sure that the learned models could be quickly and easily updated on a mobile device with a very small memory footprint. We used the WinMine toolkit (Chickering 2002), which makes splits in the decision trees using greedy search guided by a Bayesian scoring function (Chickering et al. 1997). Because the decision trees represent conditional probability distributions over discrete variables (either a predicate or an argument), the WinMine Toolkit learns multinomial distributions in the leaves of these trees. Intuitively, the tree-growing algorithm works by recursively identifying the feature for which knowing its value reduces most the uncertainty in the target variable.

As specified previously, the predicted variable G is the next user goal; and we decompose that into models for predicting the next predicate and the next argument of the predicate. It is important to note that although Eq. 1 applies to all predicates and arguments in the C&C application, in this paper, we only describe argument models for the contact items. We use the term *contact model* to refer to an argument model specific to the contact items. Note that contact-related commands constituted the majority of all commands made by users.

3.4.1 Feature engineering

Our pursuit of predictive user models was motivated by the intuition that the personal nature of mobile devices lends itself to statistical modeling. So, if a user calls a spouse at the end of every workday, or if a user listens to music every morning, we wanted to generate features to capture that. Whereas it seemed reasonable to assume that individual users might display personal idiosyncrasies or systematic patterns of usage for the C&C application, whether or not these patterns might be consistent across all users remained to be seen empirically. In order to assess if systematic patterns of usage prevailed in the general population of users, we made sure that the features we engineered to capture these patterns could be generalized across all users. For

Table 2 Different types of features generated for predicting the next user goal

Device-related
IsCalendarEnabled, AreContactsEnabled, IsMediaEnabled, IsPhoneEnabled, IsStartMenu-Enabled, UserHardware {Pocket PC, Smart Phone}, NumContacts, NumAlbums, NumArtists, NumGenres, NumStartMenus
Time-related
DayOfWeek, IsWeekend, TimeOfDay Morning, Afternoon, Evening, IsWorkingHour, Month
Predicate-related
LastPredicate, MostFreqPredicate, FractOfMostFreqPredicateToRest
Contact-specific
GuessProb, NumFilledFields, HasPrefix, HasFirstName, HasMiddleName, HasLastName, Has-Suffix, HasHome, HasCell, HasWork, HasCompanyName, IsSameAsLastItem, LastCallLoca-tion {Home, Work, Mobile,...}
Periodic (referring to both predicates and contact items)
MostFreqInDay, MostFreqInHour, MostFreqIn30Min, DurSinceLast, FractOfDayMatch, FractOfHourMatch, FractToRest, FractInHour, FractIn30Min, FractOfMostFreqDayMatch-es, FractOfMostFreqHourMatches, NumSoFar, TimeMatchesDayOfLast, TimeMatchesHour OfLast

example, suppose one user calls a relative every Saturday while another user calls every Wednesday. If we only kept track of the day of the week as our feature, we could not pick out that pattern. What is needed is a *periodic* feature such as how often calls made to a particular contact occur on the same day of the week.

Table 2 lists the different types of features we used for predicting the next user goal. A full description of each feature is listed in the Appendix. For the contact model, we used all the features. For the general predicate model, we excluded contact-specific features. Note that with the periodic features, an item could be a particular predicate (e.g., *Battery*) or contact item (e.g., *Max Chickering*). Hence, for the predicate model, which defined a distribution over 17 possible predicates, we created periodic features for each of the predicates. Because we had 14 generic periodic features, this meant that overall we had well over 200 periodic features for the predicate model.

Although both the predicate and contact models could have benefited from WinCE OS-level information, such as what applications were currently running, and application-level information, such as how many phone calls, emails and text messages had been received from particular contact items, we only had access to the commands spoken by the users and a few pieces of device data. As such, most of the features listed in Table 2 are based on time-stamp information for spoken commands. We plan to extend our logging efforts in the future.

4 Building and evaluating the user models

In this section, we present and analyze the models we learned for predicting the next user goal, which in turn entails both the predicate model and the argument model for contact items.

4.1 Cross-validation

In order to obtain variance terms for the results, we performed 10-fold cross validation. Because the order and time in which commands are issued by different users

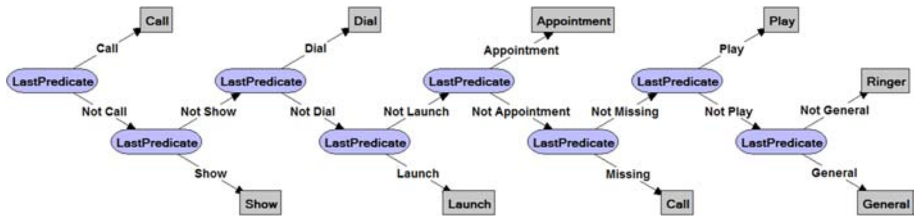

Fig. 2 A decision tree for predicting the next predicate. The leaves indicate what the most likely predicate is, given the path down the tree

matters for the features, we had to partition the dataset by individual users, not spoken commands. Furthermore, individual users had varying numbers of contacts in their contact lists. Because it was not possible to know and control in advance the number of contacts users called from their contact lists, we created the 10-folds based on the number of commands submitted by individual users. As such, we were only able to approximate 10% of the data for each fold. We revisit the issue of partitioning in Sect. 6.

4.2 Predicate model

The predicate model defines a distribution over the children of the root node in the tree of user goals shown in Fig. 1. We found that all the decision trees for the different folds contained one particular feature: the last predicate. For the fold that achieved the highest classification accuracy, the decision tree, which is shown in Fig. 2, was based solely on the last predicate. Examples for the predicates are listed in Table 1. The leaves of the decision tree in the Figure indicate what predicate is the most likely, given the path down the tree. Observing both the splits and the leaves, the tree is, to a first approximation, characterizing an equality relation which states, in short, that the next predicate is likely to be just like the last one. For example, if the last predicate was a phone call, the next predicate is most likely to be a phone call as well. In Sect. 6, we investigate this statistical relationship further.

4.2.1 Evaluation

Table 3 provides an evaluation of the predicate model in terms of both average classification accuracy and perplexity across the 10-folds. With the predicate model, classification accuracy measures how often the predicate considered to be most likely by the predicate model happens to be correct. We considered two baselines for comparison: first, the *uniform* model, in which all predicates are equally likely, and second, the *prior* model, which is based on the empirical frequencies of the predicates. For the uniform model, we calculated the theoretical values for the classification accuracy and perplexity.

With respect to classification accuracy, the prior model correctly predicted the next predicate about 43% of the time by guessing the most frequently occurring predicate. The decision tree achieved an impressive 69% average classification accuracy, which is roughly an absolute difference of 26%, using mostly the last predicate as the dominant feature. In other words, without hearing the actual command a user is speaking,

Table 3 Average classification accuracies and perplexities for the predicate models in 10-fold cross-validation

Predicate model	Class accuracy	Perplexity
Uniform	5.9%	7.13
Prior	43.2% (2.8%)	2.97 (0.62)
Decision Tree	69.2% (1.83%)	1.83 (0.36)

The parentheses indicate standard errors of the mean

69% of the time it is possible to guess what predicate will be intended by primarily considering the last predicate that was produced.

4.2.2 Perplexity

In order to assess the quality of a language modeling technique, perplexity is often used. Perplexity roughly estimates the amount by which a language model reduces uncertainty about the next item (Manning and Schütze 1999). The better the model, the lower the perplexity. More precisely, perplexity is computed as:

$$perplexity(P; P_M) = 2^{-\sum_D P(D)\cdot\log P_M(D)}$$

where the exponent is the *cross-entropy* of the true but unknown data distribution P with regard to the model distribution P_M. The average log-likelihood of the data given the model can be used as an empirical estimate of cross-entropy (Rosenfeld 2000).

Because the domain is quite small, with just 17 possible predicates, if we assume they are all equally likely, the perplexity of the domain is just 7.13. The prior model is quite informative and reduces the perplexity by more than a half. Using the decision tree reduces the perplexity even further to just 1.83, which is about a quarter of the theoretical perplexity.

4.3 Contact-argument model

From the previous section, we see that the perplexity of the domain for the predicate model is relatively small. A more challenging prediction problem, due to the fact that users in our data had anywhere from 5 to over 2,000 contacts on their mobile devices, is to determine which contact a user is likely to request. In this section, we describe how we built decision trees that could predict the intended contact item of the *Call* or *Show* predicates.

Here we consider models for the second term of Eq. 1:

$$p(A_{\pi(g)} = a(g)|\Pi = \pi(g), \boldsymbol{F} = \boldsymbol{f})$$

restricted to those situations in which $\pi(g)$ is either a *Call* or *Show* predicate. We can simplify notation here by using the variable C instead of $A_{\pi(g)}$. Furthermore, we leave implicit the conditioning on the predicate $\Pi = \pi(g)$, leaving

$$p(C = c(g)|\boldsymbol{F} = \boldsymbol{f}) \tag{2}$$

where $c(g)$ is the contact associated with the user goal.

In prior research (Yu et al. 2003), statistical models maintained an explicit distribution $p(C = c(g))$ (without input features) over the contact list of the user. We took an alternative approach where, for any set of contacts, we created a number of *binary* variables that indicated whether or not each specific contact was the intended $c(g)$. We use $B_{C=c}$ to denote the binary variable corresponding to $C = c$. For each contact, we also created a set of input features that could be used to predict the corresponding binary variable. These input features are generic properties of the contact that are "instantiated". We call these input features *contact-specific features*. We list many of the contact-specific features in Table 2.

As an example, suppose there are three contacts in a user's contact list: Joe, Steve and Mary. In this case, we would have three binary variables $B_{c(g)=Joe}$, $B_{c(g)=Steve}$ and $B_{c(g)=Mary}$, where $B_{c(g)=Joe}$ is defined to be *true* if Joe is the intended contact of the next goal (and similarly for the other two). To predict $B_{c(g)=Joe}$, we would instantiate the contact-specific feature *HasCell* for Joe, which indicates whether or not there is a cell phone number for Joe. When predicting $B_{c(g)=Mary}$, the corresponding value for the *HasCell* feature will instead indicate whether or not there is a cell phone number for *Mary*.

In our approach to building a contact-argument model, we learn a *single* model that predicts every binary B variable, using as training data all the contacts for every user. In other words, we learn a single generic contact model that applies to every specific contact in any user's list. Using $\boldsymbol{f}(c)$ to denote the contact-specific features corresponding to contact c, our model represents, for any contact c,

$$p(B_{C=c} = true|\boldsymbol{f}(c))$$

To use this model to construct $p(C = c(g)|\boldsymbol{F} = \boldsymbol{f})$, we simply normalize the values of $p(B_{C=c(g)} = true|\boldsymbol{f}(c(g)))$ for all c:

$$p(C = c(g)|\boldsymbol{F} = \boldsymbol{f}) = \frac{p(B_{C=c(g)} = true|\boldsymbol{f}(c(g)))}{\sum_{c'} p(B_{C=c'} = true|\boldsymbol{f}(c'))} \tag{3}$$

Initially, we applied this model to all the contacts in a contact list. We found that by restricting our model to those *seen* contact items that had been shown or called at least once so far—and assigning the same probability to the remaining *unseen* items—the performance of the system improved. To describe this slight modification to the model, we use $\boldsymbol{s}$ for the set of seen contacts. We then model the probability $p(seen)$ that the next contact will be a member of $\boldsymbol{s}$, yielding our final contact model:

$$p(C = c(g)|\boldsymbol{F} = \boldsymbol{f}) = \begin{cases} p(seen) \cdot \dfrac{p(B_{C=c(g)} = true|\boldsymbol{f}(c(g)))}{\sum_{c' \in \boldsymbol{s}} p(B_{C=c'} = true|\boldsymbol{f}(c'))} & \text{if } c(g) \in \boldsymbol{s} \\ (1 - p(seen)) \cdot \frac{1}{\sum_{c' \notin \boldsymbol{s}}} & \text{otherwise} \end{cases} \tag{4}$$

To model $p(seen)$, or the probability of seeing any new contact, we simply update the following fraction:

$$\frac{\sum_{c \in \boldsymbol{s}} k(c) + 1}{N + 2} \tag{5}$$

where the function $k(c)$ denotes the number of times any seen contact has been called or shown so far, and N denotes the total number of requests made so far. For smoothing, we added 1 to the numerator and 2 to the denominator.

Although our approach implicitly treats each contact item as independent of the others, the benefit compared to maintaining an explicit distribution over the contact list of each user is that we could learn a single generic contact model for all users, aggregating the important features for distinguishing whether any contact is likely to be the one requested next. This allows us to have a reasonable distribution for new users for whom we do not have any data. How well this approach compares to learning a model that leverages the same feature set and maintains an explicit distribution over the specific contact list of any particular user over time remains to be validated empirically. We developed our approach primarily as a way to overcome data sparsity and to maximize use of the data.

Similar to the predicate model, we learned a decision tree for $p(B_{C=c(g)} = true|\boldsymbol{f}(c(g)))$. Figure 3 shows a portion of the decision tree model for the fold that achieved the highest classification accuracy. The first split starts with whether the given contact is the same as the last requested contact. The majority of the features on which splits were made in the decision tree were time-related or periodic. In fact, for all of the decision trees learned across the 10-folds, periodic features constituted the majority of features on which splits were made. Full descriptions of each represented feature are provided in the Appendix. Paths through the decision tree reveal a number of fairly intuitive statistical patterns. For example, the bottom two leaves in the tree state that if a user is calling a contact who happens to be the most frequently called person in the last 30 min, and the user has just tried calling that person less than a 0.79 min (47 s) ago, then it is either 90% or 75% certain that the user will try to call that contact again, depending on how frequently that contact has been called within the hour. These two paths suggest that if users fail to reach someone, they very frequently try and try again.

4.3.1 Evaluation

In this section, we evaluate the contact model in terms of both classification accuracy and perplexity. For the purpose of calculating classification accuracy, whenever there is not a unique highest-probable contact, we choose from the set of highest-probable contacts uniformly at random. An example of when this occurs is the first time any contact command is issued. In this case, all unseen contacts are equally (and maximally) likely, and there are no seen contacts.

Similar to the predicate model, we compared the contact model against two baselines. In the *uniform* model, for assessing classification accuracy, we simply picked a contact at random from the contact list of the user at the time of the command. For assessing perplexity, we used the uniform probability over the contact list. On average, the test cases for the 10-folds had 274 contacts. The highest number of contacts for any individual user was 2,700. As another baseline, the *prior* model kept track of contact requests on an ongoing basis and selected the most frequently occurring contact. If no contact had been seen, similar to the contact model, we selected a contact at random from the unseen contacts for assessing classification accuracy.

Table 4 displays the average classification accuracies and perplexities for the different contact models. With respect to classification accuracy, the prior indicates that if we just select the most popular contact as the intended target, we would be right on average 30% of the time. Applying the decision tree according to the equations described above lifts the average classification accuracy up to 40%. Following Dietterich (1998), we tested to see if the decision tree model performed significantly better than the

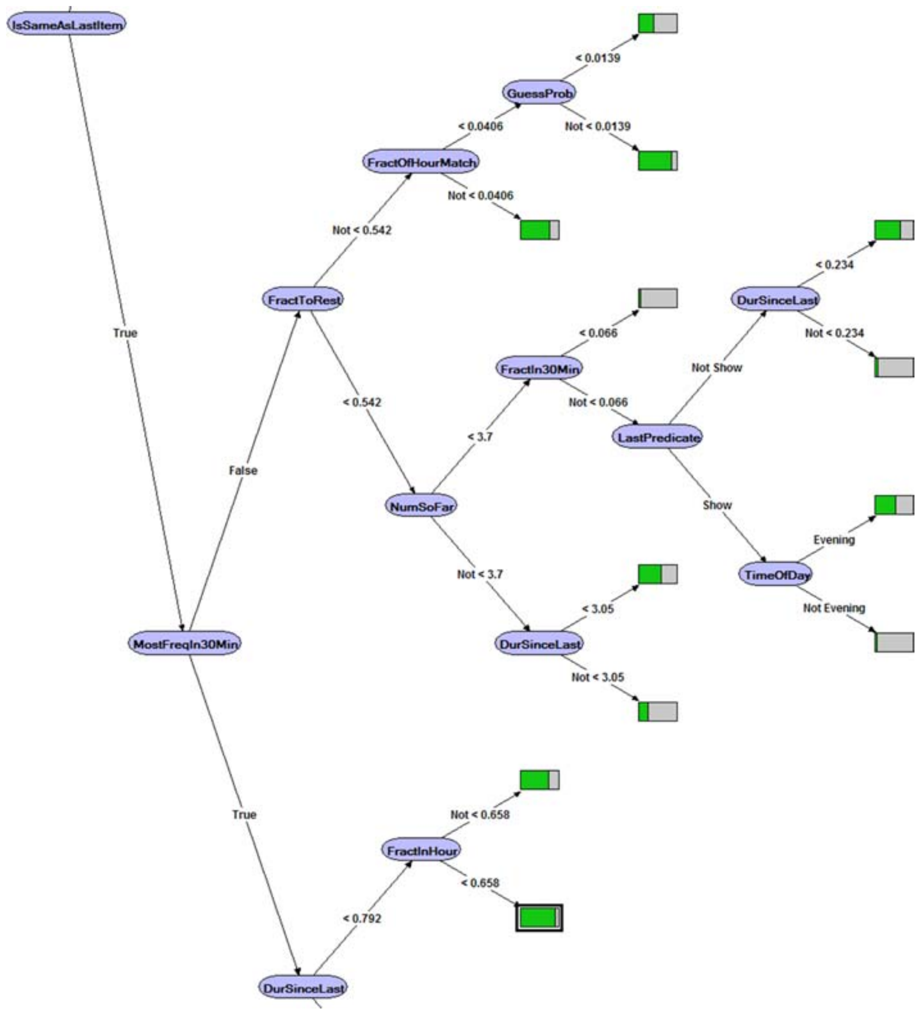

Fig. 3 A portion of a decision tree for predicting whether a given contact is likely to be the next requested contact. The decision tree starts with whether the given contact is the same as the last requested contact. The leaves display likelihood as a histogram bar weight

prior model using McNemar's test, which has a lower Type I error than a paired *t*-test for evaluating cross-validation results. The decision tree significantly outperformed the prior ($\chi^2(1) = 83.3, p < 0.001$). In short, even without hearing the contact name,

Table 4 Average classification accuracies and perplexities for the contact-argument models in 10-fold cross-validation

Contact model	Class accuracy	Perplexity
Uniform	0.1% (0.03%)	59.31 (18.76)
Prior	30.7% (9.72%)	32.54 (10.29)
Decision Tree	40.47% (12.8%)	10.84 (3.43)

The parentheses indicate standard errors of the mean it is still possible to guess the intended contact about 40% of the time. Note that this result was obtained with a general model that applies to all users.

The average perplexity for the uniform model is about twice as large as the prior model. Because the most popular contact may not always be the intended target, we calculated the probability of any contact in the prior model using just counts as follows:

$$\frac{Count(c_i)+1}{N+C} \tag{6}$$

where $Count(c_i)$ denotes the number of requests for the ith contact, N again denotes the total number of contact requests made so far and C denotes the total number of contacts on the device. If we use a decision tree, we can reduce the perplexity of the prior model to about a third of its value.

5 Language modeling

Although the user models for predicting the next predicate and contact achieved impressive classification accuracies and perplexity reductions, our primary objective is to improve C&C speech recognition. We now explain our approach to applying predictive user models for language modeling and we evaluate the practical performance of all models.

5.1 Approach

Language modeling for speech recognition is typically focused on obtaining a distribution over the possible words. In a C&C setting, however, what matters most is task completion. The task-completion goal is to identify the correct value of the goal variable G given an acoustic signal η. Using a Bayesian framework, our goal is to identify the value g of variable G that maximizes:

$$p(g|\eta,\boldsymbol{f}) = \alpha \cdot p(\eta|g)p(g|\boldsymbol{f}) \tag{7}$$

where α is a normalization constant, and we assume that the acoustic signal is independent of the input features once we know the user goal. Using the Microsoft Speech API (SAPI), which does not use our input features, we can extract the posterior probability distribution $p_{Uniform}(g|\eta)$ under the (not appropriate) assumption that every value g is equally likely apriori:

$$p_{Uniform}(g|\eta) = \alpha' \cdot p(\eta|g) \tag{8}$$

where α' is another normalization constant. From Eqs. 7 and 8, we can obtain the posterior of interest by simply multiplying $p_{Uniform}$ from SAPI by our language model and then renormalizing:

$$p(g|\eta,\boldsymbol{f}) \propto p_{Uniform}(g|\eta) \cdot p(g|\boldsymbol{f}) \tag{9}$$

5.2 Simulation environment

From a practical standpoint, classification accuracy and perplexity mean very little if they do not ultimately improve task completion. To assess how well our approach to language modeling could improve speech recognition, we tested our approach using 10-fold cross-validation in a simulation environment.

The simulation environment proceeded as follows. To simulate the same type of recognition afforded on a Pocket PC or Smart Phone device, we used a desktop version of the WinCE speech engine and loaded the same Voice Command CFG grammars. It is important to note that the desktop version is not the engine that is commercially shipped and optimized for particular devices, but rather one that is intended for research purposes, providing access to probabilities in the word lattice. For each user in the hold-out dataset, we processed each sound file, or command, along with its transcription and associated device data in chronological order. Upon submitting the sound file to SAPI, we obtained the first term in Eq. 9 and then multiplied that by the probability distribution from our user models. After deriving a distribution over all possible user goals, we checked to see if the top user goal matched the correct goal in the transcription. Finally, we observed the correct goal in order to update the data structures that kept track of the features for the models. This was done so that we could evaluate the effectiveness of our approach assuming that the features we kept track of were correct.

Because only two of the 17 possible predicates have contact items as arguments, we divided the evaluation into two tasks. In the *predicate model task*, the objective was to predict the next predicate, irrespective of the argument. In other words, if the true next user goal was $\langle Call, John\rangle$, and the model predicted the predicate *Call*, then the prediction would be correct. In the *goal model task*, the objective was to predict not only the next predicate, but also the next contact item, given that the predicate was *Call* or *Show*.

Overall, the goal model task is harder than the actual C&C application user scenario, even though start menu items and media items were not included as argument models. This follows because (1) we did not allow for disambiguation, and (2) we did not apply a rejection threshold. Normally, in the application, when a command was found to be ambiguous (e.g., “Call John” may refer to more than one contact) or the confidence score fell below a threshold for executing actions, (e.g., without-of-grammar commands) Voice Command would engage in a confirmation dialogue, and very often discover the intended user goal.

5.3 Evaluation

Because we are interested in task completion, we report the task error rates of the user models and their relative reduction in error rates for the two different tasks. The relative reduction in error rate, or *relative reduction* for short, is measured with respect to the base C&C application, which serves as our baseline.

5.3.1 Predicate model task

Table 5 shows the average error rates and the relative reductions for the prior model and the decision tree. The prior model achieves about 29% relative reduction, which is impressive given that it is based on just the frequencies of the predicates in the training data. The relative reduction for the decision tree, however, is higher still at 51%.

Because roughly 20% of the data contained one type of noise or another in the background, as transcribed by a human annotator, we separated the results into

Table 5 Average task error rate and relative reduction in error rate, as compared with the base (Uniform) system, for the predicate model task

Predicate model	Error rate	Relative reduction
Uniform	30.1% (1.7%)	0.0%
Prior	21.7% (2.3%)	28.9% (5.3%)
Decision Tree	14.9% (1.4%)	51.0% (3.3%)

The parentheses indicate standard errors of the mean

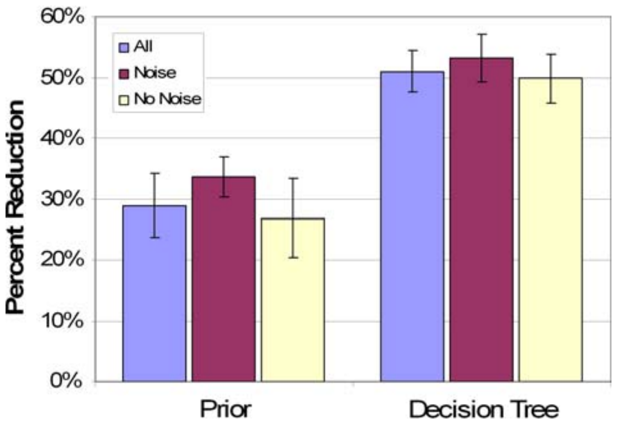

Fig. 4 Average relative reduction in error rate for the predicate model task broken down by whether or not the produced command contained noise

whether or not the spoken command had noise. Figure 4 displays a breakdown of the average relative reduction for both the prior model and decision tree. Interestingly, the prior model does slightly better in noisy conditions, jumping up to 33.7% relative reduction with a fairly small standard error of the mean of 3%. In cases in which no noise was present in the background, the relative reduction is lower with greater variability. For the decision tree, the difference in relative reduction between those commands with and without noise is just 7%, with about 4% standard error of the mean. This suggests that the decision tree may be a more robust model for improving speech recognition under any auditory condition—that is, the decision tree can be deployed without assuming anything about noise in the application environment.

5.3.2 Goal model task

Table 6 displays the error rates and the relative reductions for the goal model task where decision trees were used for both the predicate and contact-argument models. Overall, using decision trees to predict the next goal reduces the task error rate on average by 26%, compared to not using a user model. Similar to the performance of the decision tree in the predicate model task, the goal model does not increase or decrease relative reduction much when background noise is present in the produced command. Although the average relative reduction is roughly half of the average relative reduction for the predicate model task, it must be remembered that 26% rel-

Table 6 Average task error rate and relative reduction in error rate, as compared with the base (Uniform) system, for the goal model task

		Error rate	Relative reduction
All	Uniform	35.1% (1.8%)	0.0%
	Decision Tree	25.7% (1.3%)	26.3% (2.3%)
Noise	Uniform	55.1% (2.8%)	0.0%
	Decision Tree	38.8% (2.7%)	29.3% (4.0%)
No noise	Uniform	29.8% (1.8%)	0.0%
	Decision Tree	22.3% (1.1%)	24.4% (2.3%)

The results are broken down into whether or not the produced command contained noise. The parentheses indicate standard errors of the mean

ative reduction is achieved over the entire population of users in the data collection without any personalized adaptation.

6 Online personalization

In evaluating the user models applied to language modeling, we were surprised to find that without any online adaptation of model parameters, we could achieve on average a 26% relative reduction in task error rate as compared with the base application. One question remaining now is how much can we reduce task error rate by adapting to users? We found that we could exploit systematic patterns of usage across the entire population of users, so we hypothesized that adapting to any idiosyncratic patterns of usage would improve the overall performance. We now describe how we tested this hypothesis and analyze the results.

6.1 Adjusting the learning rate

A disadvantage of using decision trees as our models is that unless we are willing to re-learn each tree in real-time, we are committed to the splits in those tree and we can only update the parameters at the leaves. Despite this disadvantage, we decided to evaluate how well these models would adapt by updating those parameters in an online fashion. As such, we had to decide how to set the learning rate, which defines how quickly model parameters should be updated in light of new data.

Let β denote the learning rate. As described earlier, we used multinomial distributions in the leaves of the trees; let θ denote a current probability value from such a distribution within the model. For example, θ might denote the probability that the next predicate is a call. Using an online-update algorithm, we obtain a new estimate for θ, which we denote θ', as follows:

$$\theta' = (1 - \beta) \cdot \theta + \beta \cdot I$$

where I is the indicator function for whether the state corresponding to the parameter occurred or not (e.g., whether or not the next predicate was, indeed, a call). The learning rate β can be calculated as:

$$\beta = \frac{ESS_{NEW}}{ESS_{NEW} + ESS_{OLD}}$$

where ESS_{OLD} is the equivalent sample size of the old data on which the model had been trained, and ESS_{NEW} is the equivalent sample size of the new data that is being observed. The equivalent sample size can be considered a type of "weight" on new and old data. For example, if ESS_{OLD} is set to 1, then that specifies that the parameter that is being updated should be treated as if it were learned from one training instance, even if it may have been learned over thousands of instances. Likewise, if ESS_{NEW} is set to 10, then each new instance is treated as if it were 10 instances. For our experiments, we fixed ESS_{NEW} to 1.

The learning rate can be kept constant, or it can be adjusted over time. In the experiments we conducted, we either set β to a constant in the range from 0 to 0.25, thereby adjusting ESS_{OLD}, or we used the following simple adjustment schedule, which we refer to as the *cumulative* learning rate:

$$\beta = \frac{1}{1+N}$$

where N is the number of individual user observations seen so far. We initialized N to 1 so that the learning rate starts at 0.5 and gradually converges to zero as N accumulates.

6.2 Evaluation

Setting the learning rates as described above, we investigated the effects of personalization on both the predicate model and goal model tasks.

6.2.1 Predicate model task

Figure 5 displays the average relative reductions for the prior and decision tree models at the different learning rates. Two important trends stand out. First, the performance of the decision tree does not seem to be influenced much by the learning rate. As such, using the cumulative learning rate resulted in the same performance as the

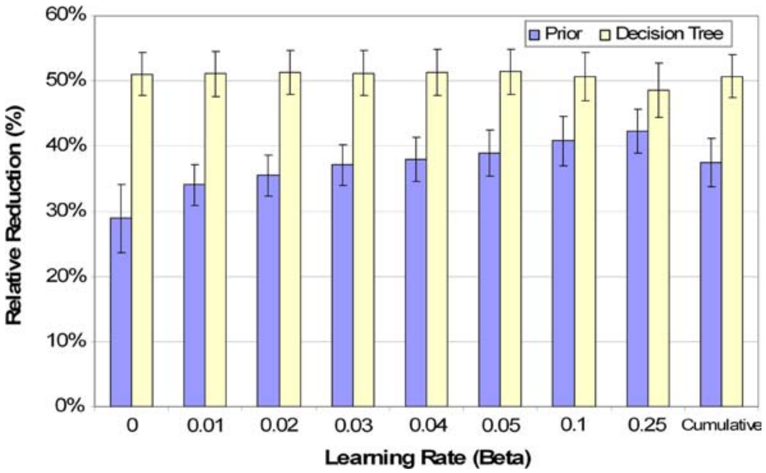

Fig. 5 Average relative reduction in task error rate of the prior and decision tree models for the predicate model task at different learning rates

fixed learning rates. Second, the prior model seems to be continually improving as the learning rate is set higher and higher. To better understand what is behind both of these trends, it is instructive to reconsider from Sect. 4.2 what the most predictive feature was for the decision tree: namely, the last predicate. If the dominant statistical trend in the data happens to be that the next predicate is most likely to be the same as the last predicate, then it makes sense that the decision tree would learn such an equality relation. Furthermore, it makes sense that as the learning rate increases, the prior model would continue to improve because a learning rate of 1 essentially means that model is throwing away all past data except the last observation. This also explains why the cumulative learning rate, which decreases the learning rate over time, performed about as well as any learning rate less that 0.1. In order to get a sense of what the upper bound on performance would be, we took the entire data set and calculated the classification accuracy of predicting the next predicate based on the last predicate. The classification accuracy was 70.4%. Recall from Table 3 that the average classification accuracy for the decision tree was 69.2% (1.83%). The fact that the average relative reduction for the decision tree remains relatively constant across different learning rates suggests that even with personalization, users are still doing what everyone else is on their mobile devices; namely, performing whatever predicate they did last. In sum, for the predicate model task, personalization did not improve performance much, primarily because all users tend to follow the same pattern.

6.2.2 Goal model task

Table 7 displays the average perplexity and relative reductions for the goal model task using decision trees for both the predicate and contact-argument models. A learning rate of zero indicates no learning and corresponds to the result in the Sect. 5.3.2. Although personalization at a learning rate of about 0.02 improves the relative reduction lightly, it is not statistically significant. Learning too quickly at rates higher than 0.03 actually results in higher perplexity than no learning. Furthermore, reduction in average perplexity is fairly small, even at the best learning rates.

6.3 Longer sequences

Even if all users are performing whatever predicate they did last, that alone does not explain why personalization did not improve performance on the goal model

Table 7 Average perplexity and relative reduction in task error rate reductions using decision trees for the goal model task at different learning rates

Learning rate	Perplexity	Relative reduction
0	8.06 (0.60)	26.3% (2.3%)
0.01	7.87 (0.59)	27.0% (2.5%)
0.02	7.86 (0.57)	27.2% (2.7%)
0.03	8.02 (0.59)	26.8% (2.6%)
0.04	8.16 (0.59)	26.4% (2.7%)
0.05	8.31 (0.59)	26.4% (2.6%)
0.1	9.37 (0.70)	25.3% (2.8%)
0.25	16.44 (2.56)	20.1% (3.4%)
Cumulative	8.09 (0.62)	26.7% (2.7%)

The parentheses indicate standard errors of the mean. Zero indicates no learning

Table 8 Average number of commands and contacts for each of the 10-fold cross-validation test sets

k-fold	Avg number of commands	Avg number of contacts
1	23.09	270.77
2	26.32	138.58
3	37.85	285.69
4	22.43	324.04
5	21.39	191.87
6	24.50	181.00
7	30.69	197.50
8	23.63	264.81
9	48.33	508.75
10	22.12	384.81
Long	116.92	589.85

The long test set represents 20% of the data which was hand-selected for its large numbers of commands

task. After all, adapting the contact-argument model to whatever behavior individual users exhibit for calling and showing contacts should result in some improvement. In order to better understand why online personalization did not significantly improve performance, we examined the distribution of commands submitted by users.

Table 8 displays the average number of commands submitted by users in the 10-fold cross-validation test sets. Recall that the 10-folds were generated by approximately separating the data based on individual users so that the commands for any user would stay together. This was done because the features were dependent on the time and order in which the commands were issued. For any *k*-fold test set, while some users contributed many commands to the data collection effort, others contributed less. The highest average number of commands was just 38. Across the entire dataset, the median number of commands was only 14.5.

To see if having more personal data on which to adapt could improve performance, we took all the users that had contributed 77 or more commands and separated them out as a test set, which we called the *long* test set. Those users, 13 in all, constituted roughly 30% of all spoken commands. As shown in Table 8, the average number of commands for the long test set was significantly higher at 116 commands. Interestingly, the long test set also had larger numbers of contacts in their contact list, though that number does not reflect the number of seen contacts—that is, those that users actually call or show.

After separating out the long test set, we then learned decision trees for both the predicate and contact-argument models on the rest of the data. The decision tree for the predicate model split on just the last predicate similar to the model shown in Fig. 2. The decision tree for the contact-argument model was also similar to the 10-fold models, and did not stand out in any particular way. Table 9 shows the perplexity and relative reduction for the decision trees on the goal model task. Whereas setting the learning rate to 0.04 had the highest perplexity reduction, setting it to 0.02 had the highest improvement in relative reduction, which is ultimately what we care about most.

Figure 6 shows the difference in relative reduction between the goal model task with no personalization and personalization with the learning rate set to 0.02. In order to obtain standard errors of the mean, we tested the decision trees on each of the 13 users that made up the long test set. Having optimized the learning rate to see

Table 9 Perplexity and relative reduction in task error rate for the goal model task using decision trees learned on users who contributed less than 77 commands and tested on those who contributed at least 77 commands

Learning rate	Perplexity	Relative reduction
0	11.6	31.0%
0.01	9.8	33.8%
0.02	9.4	35.7%
0.03	9.2	35.0%
0.04	9.1	34.8%
0.05	9.1	33.8%
0.1	9.2	32.3%
0.25	12.4	24.3%
Cumulative	10.1	34.8%

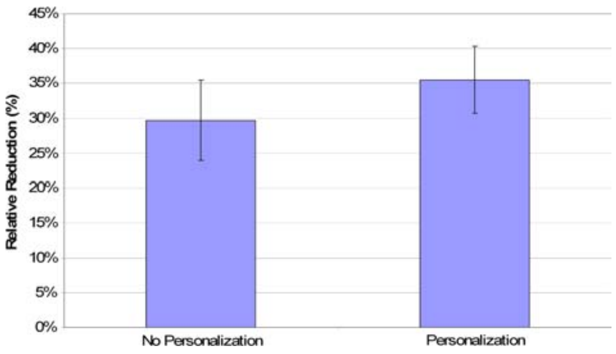

Fig. 6 Average relative reduction in task error rate for the goal model task on the users in the long test set

if personalization could be beneficial, we found that personalization did significantly improve relative reduction by an additional 5% according to McNemar's test ($\chi^2(1) = 10.41, p < 0.01$). In sum, in personalizing to users over longer sequences of commands, it is possible to improve relative reduction in task error rate.

7 Conclusion

When we first embarked on applying user models to language modeling, we originally did not expect much improvement in C&C recognition performance for predictive models that were generalized across all users. We were pleasantly surprised to find that we could achieve on average 26% relative reduction in task error rate compared to the base system using decision trees for the predicate and argument models. Furthermore, we found that personalization via online updating of model parameters to individual user data can significantly improve relative reduction up to 5% for users in the dataset that contributed many commands.

Given that the predictive user models we learned for C&C achieved such high classification accuracies without knowing anything about the actual spoken command—on average 69% for the predicate model and 40% for the contact-argument model, for future research, we plan to investigate alternative approaches to language modeling leveraging the user models. For online personalization, because users can always change their preference behaviors over time, we need to handle issues related to concept drift (Webb et al. 2001) and are considering ways to adjust the learning rate to adapt to these kinds of changes (Widmer and Kubat 1996).

Another promising area to explore is building alternative classes of models with a small memory footprint that can be easily updated. For online learning, we were committed to the splits in our decision trees and only updated the leaves. Another interesting avenue to pursue would be to incorporate online structure learning, whether it be for decision trees or other models, along with online parameter updating. It will be interesting to weigh the gain in performance of online structure learning, if any, against the computational cost of performing that model search on a mobile device.

Finally, whereas our experimental results are promising, they were performed in a simulation environment where the "user" was not aware of the adaptation of the system. It will be interesting to conduct a usability study similar to Oviatt et al. (1998) to see if user behavior changes as a result of perceived adaptation or lack thereof. For example, users may cease to use a full name to distinguish between contacts that share the same first name because they know that the system has adapted to whom they call the most; that is, they know that when they say "Call John", the system will know which John they mean.

Acknowledgments

We thank the Microsoft Voice Command team for their support and data collection. We also thank our reviewers, and in particular, one anonymous reviewer who went above and beyond in providing very helpful and constructive criticism.

Appendix

Following are full descriptions for each of the features listed in Table 2.

A Device-related

1. IsCalendarEnabled: Whether commands for accessing the user's calendar has been enabled in the Voice Command control panel.
2. AreContactsEnabled: Whether commands for accessing the user's contacts has been enabled in the Voice Command control panel.
3. IsMediaEnabled: Whether commands for accessing the user's media library has been enabled in the Voice Command control panel.
4. IsPhoneEnabled: Whether commands for calling contacts has been enabled in the Voice Command control panel.
5. IsStartMenuEnabled: Whether commands for launching start menu applications has been enabled in the Voice Command control panel.
6. UserHardware: Whether the device is a Pocket PC or Smart Phone.
7. NumContacts: Number of contacts in the user's contacts list.
8. NumAlbums: Number of albums in the user's media library.
9. NumArtists: Number of artists in the user's media library.
10. NumGenres: Number of genres in the user's media library.
11. NumStartMenus: Number of start menu applications installed on the device.

B Time-related

1. DayOfWeek: Monday, Tuesday, etc.
2. IsWeekend: Whether the time of the command was either Saturday or Sunday.
3. TimeOfDay: Whether the time of the command was Morning, Afternoon, or Evening.
4. IsWorkingHour: Whether the time of the command was between 8 AM and 7 PM.
5. Month: January, February, etc.

C Predicate-related

1. LastPredicate: Predicate of the last user command.
2. MostFreqPredicate: Most frequently occurring predicate so far.
3. FractOfMostFreqPredicateToRest: Fraction of the most frequently occurring predicate as compared to all other occurring predicates.

D Contact-specific

1. GuessProb: Probability of randomly guessing a contact in the contact list.
2. NumFilledFields: Number of fields such as Prefix, First Name, Last Name, etc. that have been filled out for the contact.
3. HasPrefix: Whether a prefix, such as Dr. or Mr. has been listed for the contact.
4. HasFirstName: Whether a first name has been submitted.
5. HasMiddleName: Whether a middle name has been submitted.
6. HasLastName: Whether a last name has been submitted.
7. HasSuffix: Whether a suffix, such as Jr., has been submitted.
8. HasHome: Whether a home number has been listed for the contact.
9. HasCell: Whether a cell number has been listed for the contact.
10. HasWork: Whether a work number has been listed for the contact.
11. HasCompanyName: Whether a company name has been submitted for the contact.
12. IsSameAsLastItem: Whether the contact is the same as the last called or shown contact.
13. LastCallLocation: Specified location if any, such as Home or Work, of the last called or shown contact.

E Periodic (referring to both predicates and contact items)

1. MostFreqInDay: Whether the item is the most frequently occurring item for the current day.
2. MostFreqInHour: Whether the item is the most frequently occurring item within the past hour.
3. MostFreqIn30Min: Whether the item is the most frequently occurring item within the past thirty minutes.
4. DurSinceLast: Duration since the last occurrence of the item.

5. FractOfDayMatch: How often the current day in which the item is being requested matches all previous occurrences of the item.
6. FractOfHourMatch: How often the current hour in which the item is being requested matches all previous occurrences of the item.
7. FractToRest: How often the current item being requested occurs in relation to all other items.
8. FractInHour: How often the current item being requested has occurred in relation to all other items within the past hour.
9. FractIn30Min: How often the current item being requested has occurred in relation to all other items within the past thirty minutes.
10. FractOfMostFreqDayMatches: How often the most frequently occurring item for the current day is requested in relation to other items that occurred on that day.
11. FractOfMostFreqHourMatches: How often the most frequently occurring item for the current hour is requested in relation to other items that occurred in that hour.
12. NumSoFar: Number of instances of the item so far.
13. TimeMatchesDayOfLast: Whether the day of the item being requested matches the day in which the item was last requested.
14. TimeMatchesHourOfLast: Whether the hour of the item being requested matches the hour in which the item was last requested.

References

Chickering, D., Heckerman, D., Meek, C.: A Bayesian approach to learning Bayesian networks with local structure. In: Proceedings of Thirteenth Conference on Uncertainty in Artificial Intelligence, Providence, RI, pp. 80–89. Morgan Kaufmann, 1997

Chickering, D., Paek, T.: Personalizing influence diagrams: applying online learning strategies to dialogue management. User Modeling and User-Adaped Interaction, 2005

Chickering, D.M.: The winmine toolkit. Technical Report MSR-TR-2002-103 Microsoft, Redmond, WA, 2002

Dietterich, T.G.: Approximate statistical test for comparing supervised classification learning algorithms. Neural Comput. **10**(7), 1895–1923 (1998)

Horvitz, E., Paek, T.: Harnessing models of users' goals to mediate clarification dialog in spoken language systems. In: Proceedings of the Eighth International Conference on User Modeling, pp. 3–13. Sonthofen, Germany, 2001

Horvitz, E., Shwe, M.: In pursuit of effective handsfree decision support: coupling Bayesian inference, speech understanding, and user models. In: Nineteenth Anuual Symposium on Computer Applications in Medical Care. Toward Cost-Effective Clinical Computing, 1995

Hunt, A., McGlashan, S. (eds.): Speech Recognition Grammar Specification Version 1.0, W3C Recommendation (2004) http://www.w3.org/TR/2004/REC-speech-grammar-20040316/.

Jameson, A., Klöckner, K.: User multitasking with mobile multimodal systems. In: Minker, W., Bühler, D., Dybkjær L. (eds.) Spoken Multimodal Human-Computer Dialogue in Mobile Environments, pp. 349–377. Kluwer Academic Publishers, Dordrecht (2004)

Jameson, A., Wittig, F.: Leveraging data about users in general in the learning of individual user models. In: Nebel B., (ed.) Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, pp. 1185–1192. Morgan Kaufmann, San Francisco, CA 2001

Jelinek, F.: Statistical Methods for Speech Recognition. MIT Press, Cambridge, MA (1997)

Johansson, P.: User modeling in dialog systems. Technical Report Technical Report SAR 02-2, Santa Anna IT Research, 2002

Manning, C.D., Schütze, H.: Foundations of Statistical Natural Language Processing. The MIT Press, Cambridge Massachusetts (1999)

Oviatt, S., MacEachern, M., Levow, G.: Predicting hyperarticulate speech during human-computer error resolution. Speech Commun. **24**(2), 87–110 (1998)

Paek, T., Horvitz, E.: Conversation as action under uncertainty. In: Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence, pp. 455–464. Stanford, CA, 2000

Rosenfeld, R.: Two decades of statistical language modeling: where do we go from here? In: Proc. IEEE **88**(8), 1270–1278 (2000)

Rosenfeld, R., Olsen, D., Rudnicky, A.: Universal speech interfaces. Interactions **8**(6), 34–44, (2001)

Strother, N.: Future cell phones: the big trends, 2005–2010. Technical Report IN0502105WH, In-Stat, Scottsdale, AZ, 2005

Webb, G., Pazzani, M., Billsus, D.: Machine learning for user modeling. User Model. User-Adapted Interac. **11**, 19–20 (2001)

Widmer, G., Kubat, M.: Learning in the presence of concept drift and hidden contexts. Machine Learning **23**, 69–101 (1996)

Woods, W.A.: Language processing for speech understanding. Computer Speech Processing, pp. 305–334, Prentice Hall, UK (1985)

Yu, D., Wang, K., Mahajan, M., Mau, P., Acero, A.: Improved name recognition with user modeling. In: Proceedings of the Eurospeech Conference, pp. 1229–1232. Geneva, Switzerland, 2003