Data-Driven Semantic Inference for Unconstrained Desktop Command and Control

Jerome R. Bellegarda and Kim E.A. Silverman
Spoken Language Group, Apple Computer, Inc.
Cupertino, California 95014, USA
{jerome, kimsilv}@apple.com

Abstract
At ICSLP'00, we introduced the concept of data-driven semantic inference, an approach to command and control which in principle allows for any word constructs in command/query formulation. Unconstrained word strings are mapped onto the relevant action through an automated classification relying on latent semantic analysis: as a result, it is no longer necessary for users to regenerate the exact syntax of every command. The objective of this paper is to further characterize the behavior of semantic inference, using a desktop command and control task involving 113 different actions. We consider various training scenarios of increasing scope to assess the influence of coverage on performance. Under realistic usage conditions, good classification results can be obtained at a level of coverage as low as 70%.

1. Introduction
At the last ICSLP, we introduced the concept of data-driven semantic inference, an approach to command and control which in principle allows for any word constructs in command/query formulation [1]. In contrast with usual inference engines (cf. [2]), semantic inference does not rely on formal behavioral principles extracted from a knowledge base. Instead, the domain knowledge is automatically encapsulated in a data-driven fashion within a low-dimensional vector space. Thus, semantic inference can be thought of as a way to perform (albeit elementary) “bottom-up” natural language understanding.

This is of interest in most command and control applications, because users are not naturally inclined to remember the exact syntax of every command. Since the typical (context-free) grammar used as language model is necessarily constrained in what it can support, this can lead to many utterances that are syntactically out-of-grammar, and therefore will be either rejected or misrecognized, even though they might be semantically adequate for the intended action. This behavior is especially discouraging for first-time users, as the implicit requirement to conform to a pre-defined formulation structure seems to contradict the intuitive and natural nature of speech communication. It would be highly desirable for such users to be able to (successfully) speak to the system in their own words.

The data-driven semantic inference framework is illustrated in Fig. 1. The inference component operates as the back-end of a large vocabulary transcription system. Thus, instead of a context-free grammar, a statistical n-gram is used for language modeling, much like in dictation applications. Note that the string of transcribed words produced by the recognizer may not be entirely correct. However, because of the limited domain of discourse, we can expect a reasonably low word error rate, which may not materially affect the basic meaning of the utterance. The issue now is to recognize the action intended to be taken. This is done through semantic classification, whose role is to extract the data-driven semantic representation closest in meaning to the user's utterance. The intended action is then generated as usual.

Semantic classification relies on evidence extracted from some training text by latent semantic analysis (LSA) [3]. Each supported action gives rise to a different vector (semantic anchor) in a suitable vector space. During recognition, mapping (the transcription of) a command onto an action amounts to comparing the corresponding command vector with each semantic anchor. The procedure is thus entirely data-driven.

The objective of this paper is to characterize the behavior of this approach under different training scenarios. The next two sections review LSA basics and the mechanics of semantic inference. In Section 4, we discuss a necessary extension to handle commands that are ambiguous at the word level. Finally, Section 5 reports experimental results on a desktop command and control task involving 113 different actions.

2. Latent Semantic Analysis
Let \( V \) be a vocabulary of size \( M \), and \( T \) a collection of utterances used to invoke \( N \) actions appropriate to some
domain of interest. For a given action, the subset of associated utterances will comprise many alternative wordings (such as “what time is it?” or “what’s the time?”) that all convey the meaning of that action (in this case, “display current time”). We say that this subset collectively expresses the command to invoke that action. We further refer to each utterance in the subset as an instance, and to each distinct wording as a variant, of that command. Typically, M and N are on the order of a few hundreds to a few thousands, with each command having scores of different variants. The task at hand is to define a mapping between the set T and a vector space S, whereby each command in T is represented by a vector in S [1].

We first construct a word-command matrix W associated with V and T, in a manner similar to that detailed in [3]. This (M x N) matrix fully describes, for the training corpus T, how often word wj appeared in command dj. We then perform the singular value decomposition (SVD) of W as:

\[ W \approx U S V^T, \]  

where U is the (M x R) left singular matrix with row vectors u_i (1 ≤ i ≤ M), S is the (R x R) diagonal matrix of singular values, V is the (N x R) right singular matrix with row vectors v_j (1 ≤ j ≤ N), R < min(M, N) is the order of the decomposition, and Tr denotes matrix transposition. As is well known, U^T U = V^T V = I_R, the identity matrix of order R. Thus, the column vectors of U and V each define an orthonormal basis for the space of dimension R spanned by the u_i’s and v_j’s, i.e., the space S sought. The dimension R is bounded from above by the rank of the matrix W, and from below by the amount of distortion tolerable in the decomposition.

If we project the column vectors of W (i.e., commands) onto the orthogonal basis formed by the column vectors of U, the new coordinates are given by the columns of SV^T. This means that, for 1 ≤ j ≤ N, the column vector Sv_j^T, or, equivalently, the row vector v_jS, characterizes the position of command dj in the space S [3]. Each of the N scaled vectors:

\[ v_j = v_j S = d_j^T U, \]  

is therefore the semantic anchor of the action uniquely associated with the command dj ∈ T. Since W captures the major associational structure in W, the closeness of vectors in S is determined by the overall pattern of the language used in T, as opposed to specific constructs. This tends to reveal meaningful associations (not just particular co-occurrences) between words and commands. The reader is referred to the R = 3 illustration presented in [1] for an example of such associations. In the interest of brevity, we will simply recall here that two commands whose representations in S are “close” tend to be associated with the same action, whether or not they contain the same word constructs.

3. Semantic Inference

This behavior means that a new variant, using a different wording, will still be “close” to the appropriate semantic anchor if that is otherwise consistent with the expected language structure (e.g., if the new variant would tend to invoke the corresponding desired action). In other words, the new variant is semantically related to that action, which can then be automatically inferred.

To perform semantic classification, it is thus sufficient to determine, for a given variant, which semantic anchor the variant is most closely aligned with. This requires finding a representation for all individual variants in the space S, including those not seen in the training corpus. Let us denote a variant by d_v, where the tilde symbol reflects the fact that this variant may not have been part of the training data. By construction, the feature vector d_v is a column vector of dimension M, which can be treated as if it were an additional column of the matrix W. Provided the matrices U and S do not change, the SVD expansion (1) implies:

\[ \tilde{d}_v = U S \tilde{v}_v^T, \]  

where the R-dimensional vector \( \tilde{v}_v^T \) acts as an additional column of the matrix V^T. Hence the variant vector:

\[ \tilde{v}_v = \tilde{v}_v S = \tilde{d}_v^T U, \]  

corresponds to the representation of the variant \( \tilde{d}_v \) in S. Note that this representation is consistent with (2): if an action is invoked with only one variant, the resulting variant vector reduces to the semantic anchor for that action.

In the case of a variant not seen in the training corpus, relatively mild assumptions are required for (4) to hold. Clearly, if the new variant contains language patterns which are inconsistent with those extracted from W, the SVD expansion (1) will no longer apply. Similarly, if the addition of \( \tilde{d}_v \) causes the major structural associations in W to shift in some substantial manner, U and S will no longer be valid, in which case it would be necessary to re-compute (1) to find a proper representation for \( \tilde{d}_v \). If, however, the new variant generally conforms to the rest of T, then \( \tilde{v}_v \) will be a reasonable representation for \( \tilde{d}_v \).

Once (4) is obtained, it remains to evaluate the closeness between the observed variant vector \( \tilde{v}_v \) and the semantic anchors \( v_j \) determined during training. As it turns out, an appropriate closeness measure is the cosine of the angle between the two vectors [3]:

\[ K(\tilde{v}_v, v_j) = \cos(\tilde{v}_v S, v_j S) = \frac{\tilde{v}_v S^T v_j^T}{||\tilde{v}_v S|| ||v_j S||}, \]  

for 1 ≤ j ≤ N. Classification occurs by assigning the variant vector \( \tilde{v}_v \) to the semantic anchor \( v_j \) associated with maximum closeness.

Semantic inference has a number of useful properties. Not only is there no a priori constraint on what properties the user can say, but it can generalize well from a tractably small amount of training data. For example, it can learn synonyms and apply them in new contexts. Suppose for instance that the training data contained “cancel the meeting” and “undo the meeting” as variants of one command, but only “cancel the last command” as a variant of another command. Because the variants for the former command indicate that “cancel” and “undo” are synonyms, the new variant “undo the last command” would still be correctly mapped.
4. Framework Extension

Exploiting co-occurrences between words and commands, as done above, is an instance of the so-called “bag-of-words” model, which pays no attention to the order of words in the variants. While well-suited to capture large-span (semantic) relationships between words [5], this makes it inherently unable to capitalize on the local constraints present in the language, be they syntactic or pragmatic. This is fine in applications where local constraints are discarded anyway (such as in information retrieval [4]), or for tasks such as call routing, where only the broad topic of a message is to be identified [5]. For general command and control tasks, however, this limitation may be more deleterious.

Imagine two commands that differ only in the presence of the word “not” in a crucial place. The respective vector representations could conceivably be relatively close in the LSA space, and yet have vastly different intended consequences. Worse yet, some commands may differ only through word order. Consider this actual example from the MacOS operating system:

change popup to window
change window to popup

Following conventional LSA, these two commands are mapped onto the *exact same point* in LSA space. This makes them obviously impossible to disambiguate.

We address this issue by extending the basic LSA framework via word agglomeration. The idea is to move from words and commands to word n-tuples and n-tuple commands, where each word n-tuple is the agglomeration of n successive words, and each (n-tuple) command is now expressed in terms of all the word n-tuples it contains. We then construct the associated “word-command” matrix to capture co-occurrences between word n-tuples and n-tuple commands. The rest of LSA proceeds as before, with the LSA space now comprising both word n-tuple vectors and n-tuple command vectors. Clearly, this model incorporates local constraints of span less or equal to n.

To illustrate, in the simple case n = 2, the commands in (6) now become, respectively:

```plaintext
# change change:popup popup; to: window window; 
# change: window window; to: popup popup;
```

where the symbol '#' represents a sentence boundary marker, and ':' is used to separate the two components of each 2-tuple. The effect of this agglomeration is obvious. Whereas at the word level the two commands (6) totally overlapped, at the 2-tuple level they only have the rather generic “change” in common, a fairly minimal overlap. It has thus become possible to distinguish between the two in the new LSA space.

Word agglomeration does not impact the command dimension of the co-occurrence matrix, but it does increase the word dimension, by a factor theoretically up to \( m^{n-1} \). The choice of \( n \) therefore centers around the following trade-off: large enough to encapsulate meaningful local constraints, and small enough to preserve generalization properties and keep the problem computationally tractable. In practice, the familiar choice \( n \leq 3 \) is compatible with the typical command and control situation.

5. Experimental Results

Experiments were conducted using 113 actions from the “Speakable Items” desktop command and control task defined on the MacOS operating system. For each action, we collected about 5 variants of the associated canonical command, from 10 different speakers. This produced a total of 4,755 instances, comprising 2,676 distinct variants and 2,079 instances arising from the repetition of (some of) these variants. A histogram of all variant instances collected is given in Fig. 2.

This figure shows a bi-modal distribution, skewed in favor of low frequency occurrences. Some variants are universally accepted across the 10 speakers, and thus tend to be used repeatedly (right-hand side of Fig. 2). This is primarily the case for the original 113 canonical commands, which appeared in all the prompts for variants and therefore were known to all speakers. (In fact, they were the natural default choice when a user could not think of enough variations.) On the other hand, the vast majority of the 2503 remaining distinct variants are highly idiosyncratic, and appear very infrequently in the training corpus (left-hand side of Fig. 2). This underscores a surprisingly small degree of overlap between speakers in the way they choose to express the various commands.

To evaluate the influence of training size on performance, we randomly split the above training data into concentric subsets of increasing size, starting with a baseline set of 2,000 variant instances and gradually adding more instances in increments of approximately 100. This led to 17 different training subsets, all containing variants for each of the 113 “Speakable Items” actions. However, in actual usage, people often do extend the set of pre-defined “Speakable Items” with additional actions. To simulate the introduction of new actions in the above increments, we therefore shuffled the training subsets slightly, to ensure that the baseline set of 2,000 instances covered only 100 of the 113 actions, the next subset covered one more of these actions, and so on. This led to our final training set up, whose coverage characteristics are summarized in Fig. 3.

In Fig. 3, three types of coverage are distinguished, to reflect the fraction of the 113 actions, 2,676 distinct
variants, and 4,755 variant instances represented in each training subset. As just discussed, only the last 4 of the 17 subsets cover the full 113 actions, to take into account realistic usage patterns. Also note that, on average, the number of distinct variants grows faster than the number of variant instances, which is due to the relatively small degree of overlap between speakers. In other words, there is a non-negligible probability that the next user will come up with a previously unseen variant.

In previous work [1], the test suite was taken to be the data provided by two additional speakers, who had not seen those training variables. The adjustments were made. In hindsight, this was largely, but not completely, satisfactory: because of their status as natural default, some of the original 111 canonical commands may have been over-represented in this test data. To avoid such possibilities, it is better advised to ensure that the distribution of variants in the test set conforms to the distribution plotted in Fig. 2. So this time we massaged the data provided by the two additional speakers to create a 332-variant test set, where each action is represented by exactly three distinct variants which conform to this distribution. We then had the two speakers re-record this "canonical" test set. We think that this more accurately reflects the level of performance that can be expected of the approach.

As in [1], the objective was to use semantic inference to classify each test command variant against the appropriate action. From the results of [1], we adopted extended LSA with word triplet agglomeration (n = 3). For each training subset, two conditions were considered: (i) using R = 100 in all cases for the order of the decomposition; (ii) using the maximum R possible for each training set, i.e., the total number of distinct actions covered at that size (100 ≤ R ≤ 113). The resulting average classification error rates are reported in Fig. 4.

In both cases the error curve shows an injection point for an instance coverage of about 0.7. Given the roughly linear nature of coverage growth (cf, Fig. 3), we conclude that semantic inference requires a sufficient amount of training data to achieve an acceptable level of performance, and, in particular, near-complete action coverage. This is not surprising, since a misclassification is guaranteed to occur when the corresponding semantic anchor is not present in the LSA space. On the other hand, performance is less sensitive to variant coverage, indicating that semantic inference does indeed generalize well to new contexts. In particular, from Figs. 3 and 4, we observe that misclassification rates inferior to 5% can be obtained with a variant coverage as low as 70%.

6. Conclusions

Semantic inference makes it possible to relax some of the interaction constraints present in traditional command and control implementations. By allowing users more flexibility in expressing the desired command/query, it tends to reduce cognitive load and thereby enhance user satisfaction [5]. The underlying framework is latent semantic analysis. Each command variant is mapped onto an action by comparing the corresponding command vector with each semantic anchor, using the closeness measure (3) in the LSA space.

Semantic inference requires a sufficient amount of training data to achieve an acceptable level of performance. In particular, complete action coverage is required to avoid systematic errors, which in practice requires recompacting the LSA space each time a new action is added. On the other hand, classification results appear to be robust to relatively low variant coverage, as expected from the semantic nature of the approach.

7. References