Interactive Machine Learning Techniques for Improving SLU Models

Lee Begeja
Bernard Renger
AT&T Labs-Research
180 Park Ave
Florham Park, NJ 07932
{lee,renger}
@research.att.com

David Gibbon
Zhu Liu
Behzad Shahraray
AT&T Labs-Research
200 Laurel Ave S
Middletown, NJ 07748
{dcg, zliu, behzad}
@research.att.com

Abstract

Spoken language understanding is a critical component of automated customer service applications. Creating effective SLU models is inherently a data driven process and requires considerable human intervention. We describe an interactive system for speech data mining. Using data visualization and interactive speech analysis, our system allows a User Experience (UE) expert to browse and understand data variability quickly. Supervised machine learning techniques are used to capture knowledge from the UE expert. This captured knowledge is used to build an initial SLU model, an annotation guide, and a training and testing system for the labelers. Our goal is to shorten the time to market by increasing the efficiency of the process and to improve the quality of the call types, the call routing, and the overall application.

1 Introduction

The use of spoken dialogue systems to automate services in call centers is continually expanding. In one such system, unconstrained speech recognition is used in a limited domain to direct call traffic in customer call centers (Gorin et al, 1997). The challenge in this environment is not only the accuracy of the speech recognition but more importantly, the knowledge and understanding of how the customer request is mapped to the business requirement.

The first step of the process is to collect utterances from customers, which are transcribed. This gives us a baseline for the types of requests (namely, the user intents) that customers make when they call a client. A UE expert working with the business customer uses either a spreadsheet or a text document to classify these calls into call types. For example,

- "I want a refund" → REFUND
- "May I speak with an operator" → GET_CUSTOMER_REP

The end result of this process is a document, the annotation guide, that describes the types of calls that may be received and how to classify them. This guide is then given to a group of "labelers" who are trained and given thousands of utterances to label. The utterances and labels are then used to create the SLU model for the application. The call flow which maps the call types to routing destinations (dialog trajectory) is finalized and the development of the dialogue application begins. After the field tests, the results are given to the UE expert, who then will refine the call types, create a new annotation guide, retrain the labelers, redo the labels and create new ones from new data and rebuild the SLU model.

Previously, this knowledge was only captured in a document and was not formalized until the SLU model was generated. Our goal in creating our system is not only to give the UE expert tools to classify the calls, but to capture and formalize the knowledge that is gained in the process and to pass it on to the labelers. We can thus automatically generate training instances and testing scenarios for the labelers, thereby creating more consistent results. Additionally, we can use the SLU model generated by our system to "pre-label" the utterances. The labelers can then view these "pre-labeled" utterances and either agree or disagree with the generated labels. This should speed up the overall labeling process.

More importantly, this knowledge capture will enable the UE expert to generate and test a SLU model as part of the process of creating the call types for the speech data. The feedback from this initial SLU test allows the UE expert to refine the call types and to improve them without having to train a group of labelers and to run a live test with customers. This results in an improved SLU model and makes it easier to find problems before deployment, thus saving time and money.

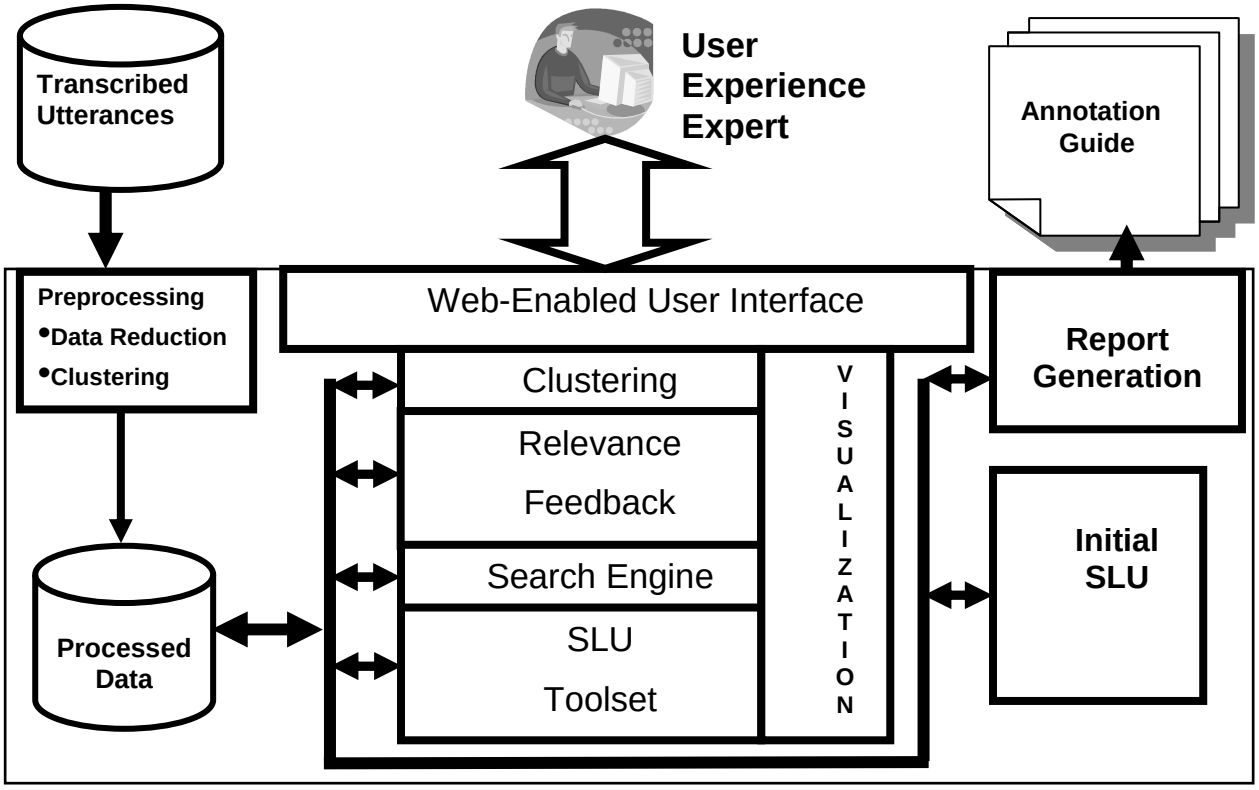

Figure 1. System Diagram

At the same time, the process is more efficient due to the increased uniformity in the way different UE experts classify calls into call type labels.

We will describe Annomate, an interactive system for speech data mining. In this system, we employ several machine learning techniques such as clustering and relevance feedback in concert with standard text searching methods. We focus on interactive dynamic techniques and visualization of the data in the context of the application.

The paper is organized as follows. The overview of the system is presented in Section 2. Section 3 briefly discusses the different components of the system. Some results are given in Section 4. Finally, in Sections 5 and 6, we give conclusions and point to some future directions.

2 System Overview

In this section, we will give a system overview and show how automation has sped up and improved the existing process. The UE expert no longer needs to keep track of utterances or call type labels in spreadsheets. Our system allows the UE expert to more easily and efficiently label collected utterances in order to automatically build a SLU model and an electronic annotation guide (see System Diagram in Figure 1). The box in Figure 1 contains the new components used in the improved and more automated process of creating SLU models and annotation guides.

After data collection, the Preprocessing steps (the data reduction and clustering steps are described in more detail below) reduce the data that the UE expert needs to work with thus saving time and money. The Processed Data, which initially only contains the transcribed utterances but later will also contain call types, is stored in an XML database which is used by the Web Interface. At this point, various components of the Web Interface are applied to create call types from utterances and the Processed Data (utterances and call types) continue to get updated as these changes are applied. These include the Clustering Tool to fine-tune the optimal clustering performance by adjusting the clustering threshold. Using this tool, the UE expert can easily browse the utterances within each cluster and compare the members of one cluster with those of its neighboring clusters. The Relevance Feedback component is implemented by the Call Type Editor Tool. This tool provides an efficient way to move utterances between two call types and to search relevant utterances for a specific call type. The Search Engine is used to search text in the utterances in order to facilitate the use of relevance feedback. It is also used to get a handle on utterance and

call type proximity using utterance and cluster distances.

After a reasonable percentage of the utterances are populated or labeled into call types, an initial SLU model can be built and tested using the SLU Toolset. Although a reduced dataset is used for labeling (see discussion on clone families and reduced dataset below), all the utterances are used when building the SLU model in order to take advantage of more data and variations in the utterances. The UE expert can iteratively refine the SLU model. If certain test utterances are being incorrectly classified or are not providing sufficient differentiability among certain call types (the SLU metric described below is used to improve call type differentiation), then the UE expert can go back and modify the problem call types (by adding utterances from other call types or by removing utterances using the Web Interface). The updated Processed Data can then be used to rebuild the SLU model and it can be retested to ensure the desired result. This initial SLU model can also be used as a guide in determining the call flow for the application.

The Reporting component of the Web Interface can automatically create the annotation guide from the Processed Data in the XML database at any time using the Annotation Guide Generation Tool. If changes are made to utterances or call types, then the annotation guide can be regenerated almost instantly. Thus, this Web Interface allows the UE expert to easily and more efficiently create the annotation guide in an automated fashion unlike the manual process that was used before.

3 Components

Many SLU systems require data collection and some form of utterance preprocessing and possibly utterance clustering. Our system uses relevance feedback and SLU tools to improve the SLU process.

3.1 Data Collection

Natural language data exists in a variety of forms such as documents, e-mails, and text chat logs. We will focus here on transcriptions of telephone conversations, and in particular, on data collected in response to the first prompt from an open dialogue system. The utterances collected are typically short phrases or single sentences, although in some cases, the caller may make several statements. It is assumed that there may be multiple intents for each utterance. We have also found that the methods presented here work well when used with the one-best transcription from a large vocabulary automatic speech recognition system instead of manual transcription.

3.2 Preprocessing

Our tools add structure to the raw collected data through a series of preprocessing steps. Utterance redundancy (and even repetition) is inherent in the collection process and this is tedious for UE experts to deal with as they examine and work with the dataset. This section describes taking the original utterance set and reducing the redundancy (using text normalization, named entity extraction, and feature extraction) and thereby the volume of data to be examined. The end product of this processing is a subset of the original utterances that represents the diversity of the input data in a concise way. Sets of identical or similar utterances are formed and one utterance is selected at random to represent each set (alternative selection methods are also possible, see the Future Work section). UE experts may choose to expand these *clone families* to view individual members, but the bulk of the interaction needs to only involve a single representative utterance from each set.

Text Normalization

There is a near continuous *degree of similarity* between utterances. At one extreme are exact text duplicates (data samples in which two different callers say the exact same thing). At the next level, utterances may differ only by transcription variants like “100” vs. “one hundred” or “$50” vs. “fifty dollars.” *Text normalization* is used to remove this variation. Moving further, utterances may differ only by the inclusion of verbal pauses or of transcription markup such as: “uh, eh, background noise.” Beyond this, for many applications it is insignificant if the utterances differ only by contraction: “I’d vs. I would” or “I wanna” vs. “I want to.” Acronym expansions can be included here: “I forgot my personal identification number” vs. “I forgot my P I N.” Up to this point it is clear that these variations are not relevant for the purposes of intent determination (but of course they are useful for training a SLU classifier). We could go further and include synonyms or synonymous phrases: “I want” vs. “I need.” Synonyms however, quickly become too powerful at data reduction, collapsing semantically distinct utterances or producing other undesirable effects (“I am in want of a doctor.”) Also, synonyms may be application specific.

Text normalization is handled by string replacement mappings using regular expressions. Note that these may be represented as context free grammars and composed with named entity extraction (see below) to perform both operations in a single step. In addition to one-to-one replacements, the normalization includes many-to-one mappings (you ← y’all, ya’ll) and many-to-null mappings (to remove noise words).

Named Entity Extraction

Utterances that differ only by an entity value should also be collapsed. For example "give me extension 12345" and "give me extension 54321" should be represented by "give me extension *extension_value*." Named entity extraction is implemented through rules encoded using context free grammars in Backus-Naur form. A library of generic grammars is available for such things as phone numbers and the library may be augmented with application-specific grammars to deal with account number formats, for example. The grammars are viewable and editable, through an interactive web interface. Note that any grammars developed or selected at this point may also be used later in the deployed application but that the named entity extraction process may also be data driven in addition to or instead of being rule based.

Feature Extraction

To perform processing such as clustering, relevance feedback, or building prototype classifiers, the utterances are represented by feature vectors. At the simplest level, individual words can be used as features (i.e., a unigram language model). In this case, a lexis or vocabulary for the corpus of utterances is formed and each word is assigned an integer index. Each utterance is then converted to a vector of indices and the subsequent processing operates on these feature vectors. Other methods for deriving features include using bi-grams or tri-grams as features, weighting features based upon the number of times a word appears in an utterance or how unusual the word is in the corpus (TF, TF-IDF), and performing word stemming (Porter, 1980). When the dataset available for training is very small (as is the case for relevance feedback) it is best to use less restrictive features to effectively amplify the training data. In this case, we have chosen to use features that are invariant to word position, word count and word morphology and we ignore noise words. With this, the following two utterances have identical feature vector representations:

- I need to check medical claim status
- I need check status of a medical claim

Note that while these features are very useful for the process of initially analyzing the data and defining call types, it is appropriate to use a different set of features when training classifiers with large amounts of data when building the SLU model to be fielded. In that case, tri-grams may be used, and stemming is not necessary since the training data will contain all of the relevant morphological variations.

Clustering

After the data reductions steps above, we use clustering as a good starting point to partition the dataset into clusters that roughly map to call types.

Clustering is grouping data based on their intrinsic similarities. After the data reduction steps described above, clustering is used as a bootstrapping process to create a reasonable set of call types.

In any clustering algorithm, we need to define the similarity (or dissimilarity, which is also called distance) between two samples, and the similarity between two clusters of samples. Specifically, the data samples in our task are call utterances. Each utterance is converted into a feature vector, which is an array of terms and their weights. The distance of two utterances is defined as the cosine distance between corresponding feature vectors. Assume **x** and **y** are two feature vectors, the distance d(**x**,**y**) between them is given by

$$d(\mathbf{x},\mathbf{y}) = 1 - \frac{\mathbf{x} \bullet \mathbf{y}}{\|\mathbf{x}\| \cdot \|\mathbf{y}\|}$$

As indicated in the previous section, there are different ways to extract a feature vector from an utterance. The options include named entity extraction, stop word removal, word stemming, N-gram on terms, and binary or TF-IDF (Term frequency – inverse document frequency) based weights. Depending on the characteristics of the applications in hand, certain combinations of these options are appropriate. For all the results presented in this paper, we applied named entity extraction, stop word removal, word stemming, and 1-gram term with binary weights to extract the feature vectors.

The cluster distance is defined as the maximum distance between any pairs of two utterances, one from each cluster. Figure 2 illustrates the definition of the cluster distance.

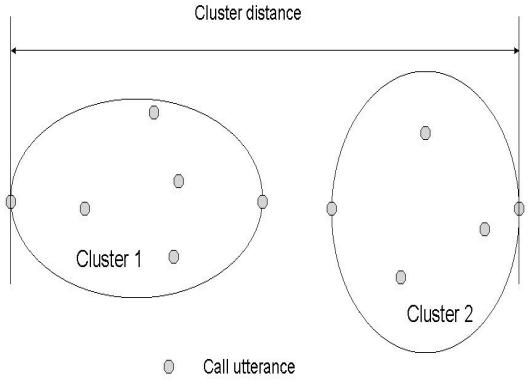

Figure 2. Illustration of Cluster Distance.

The range of utterance distance is from 0 to 1, and the range of the cluster distance is the same. When the cluster distance is 1, it means that there exists at least one pair of utterances, one from each cluster, that are totally different (sharing no common term).

The clustering algorithm we adopted is the Hierarchical Agglomerative Clustering (HAC) method. The

details of agglomerative hierarchical clustering algorithm can be found in (Jan and Dubes, 1988). The following is a brief description of the HAC procedure. Initially, each utterance is a cluster on its own. Then, for each iteration, two clusters with a minimum distance value are merged. This procedure continues until the minimum cluster distance exceeds a preset threshold. The principle of HAC is straightforward, yet the computational complexity and memory requirements may be high for large size datasets. We developed an efficient implementation of HAC by on-the-fly cluster/utterance distance computation and by keeping track of the cluster distances from neighboring clusters, such that the memory usage is effectively reduced and the speed is significantly increased.

Our goal is to partition the dataset into call types recognized by the SLU model and the clustering results provide a good starting point. It is easier to transform a set of clusters into call types than to create call types directly from a large set of flat data. Depending on the distance threshold chosen in the clustering algorithm, the clustering results may either be conservative (with small threshold) or aggressive (with large threshold). If the clustering is conservative, the utterances of one call type may be scattered into several clusters, and the UE expert has to merge these clusters to create the call type. On the other hand, if the cluster is aggressive, there may be multiple call types in one cluster, and the UE expert needs to manually split the mixture cluster into different call types. In real applications, we tend to set a relatively low threshold since it is easier to merge small homogeneous clusters than to split one big heterogeneous cluster.

3.3 Relevance Feedback

Although clustering provides a good starting point, finding all representative utterances belonging to one call type is not a trivial task. Effective data mining tools are desirable to help the UE expert speed up this manual procedure. Our solution is to provide a relevance feedback mechanism based on support vector machine (SVM) techniques for the UE expert to perform this tedious task.

Relevance feedback is a form of query-free retrieval where documents are retrieved according to a measure of relevance to given documents. In essence, a UE expert indicates to the retrieval system that it should retrieve "more documents like the ones desired, not the ones ignored." Selecting relevant documents based on UE expert's inputs is basically a classification (relevant/irrelevant) problem. We adopted support vector machine as the classifier for to two reasons: First, SVM efficiently handles high dimensional data, especially a text document with a large vocabulary. Second, SVM provides reliable performance with small amount of training data. Both advantages perfectly match the task at hand. For more details about SVM, please refer to (Vapnik, 1998; Drucker et al, 2002).

Relevance feedback is an iterative procedure. The UE expert starts with a cluster or a query result by certain keywords, and marks each utterance as either a positive or negative utterance for the working call type. The UE expert's inputs are collected by the relevance feedback engine, and they are used to build a SVM classifier that attempts to capture the essence of the call type. The SVM classifier is then applied to the rest of the utterances in the dataset, and it assigns a relevance score for each utterance. A new set of the most relevant utterances are generated and presented to the UE expert, and the second loop of relevance feedback begins. During each loop, the UE expert does not need to mark all the given utterances since the SVM is capable of building a reasonable classifier based on very few, e.g., 10, training samples. The superiority of relevance feedback is that instead of going through all the utterances one by one to create a specific call type, the UE expert only needs to check a small percentage of utterances to create a satisfactory call type.

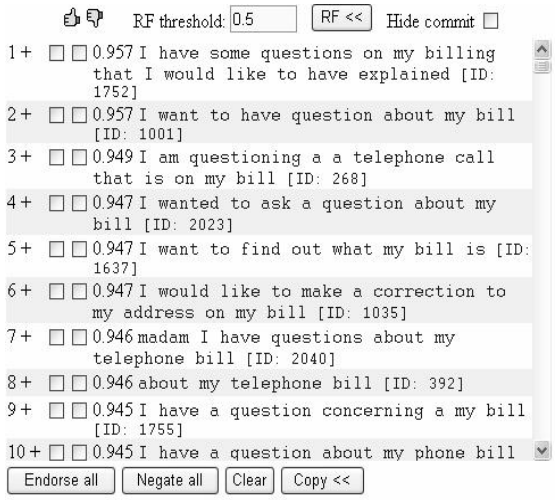

Figure 3. The Interface for Relevance Feedback.

The relevance feedback engine is implemented by the Call Type Editor Tool. This tool provides an integrated environment for the UE expert to create a variety of call types and assign relevant utterances to them. The tool provides an efficient way to move utterances between two call types and to search relevant utterances for a specific call type. The basic search function is to search a keyword or a set of keywords within the dataset and retrieve all utterances containing these search terms. The UE expert can then assign these utterances into the appropriate call types. Relevance feedback serves as an advanced searching option. Relevance feedback can be applied to the positive and negative utterances of a clus-

Industry Sector	Original Utterances	Unique Utterances	Unique Utterances after Text Normalization	Unique Utterances after Entity Extraction	Unique Utterances after Feature Extraction	Redundancy
Financial	11,623	10,021	9,670	9,165	7,929	31.8%
Healthcare	12,080	10,255	9,452	9,382	7,946	34.2%
Insurance	12,109	8,865	8,103	7,963	6,530	46.1%
Retail	10,240	4,956	4,392	4,318	3,566	65.2%

Table 1. Data Reduction Results

ter or call type or can be applied to utterances, from a search query, which are marked as positive or negative. The interface for the relevance feedback is shown in Figure 3. In the interface, the UE expert can mark the utterances as positive or negative samples. The UE expert can also control the threshold of the relevance value such that the relevance feedback engine only returns utterances with high enough relevance values. In the tool, we are using an internally developed package for learning large margin classifiers to implement the SVM classifier (Haffner et al, 2003).

3.4 SLU Toolset

The SLU toolset is based on an internally developed NLU Toolset. The underlying boosting algorithm for text classification used, BoosTexter, is described elsewhere (Freund and Schapire, 1999; Schapire and Singer, 2000; Rochery et al, 2002). We added interactive input and display capabilities via a Web interface allowing the UE expert to easily build and test SLU models.

Named entity grammars are constructed as described above. About 20% of the labeled data is set aside for testing. The remaining data is used to build the initial SLU model which is used to test the utterances set aside for testing. The UE expert can interactively test utterances typed into a Web page or can evaluate the test results of the test data. For each of the tested utterances in the test data, test logs show the classification confidence scores for each call type. The confidence scores are replaced by probability thresholds that have been computed using a logistic function. These scores are then used to calculate a simple metric which is a measure of call type differentiability. If the test utterance labeled by the UE expert is correctly classified, then the call type is the truth call type. The SLU metric is calculated as follows and it is averaged over the utterances:

- if the call type is the truth, the score is the difference (positive) between the truth probability and the next highest probability
- if the call type is not the truth, the score is the difference (negative) between the truth probability and the highest probability

This metric allows the UE expert to easily spot problem call types or those that might give potential problems in the field. It is critical that call types are easily differentiable in order to properly route the call. The UE expert can iteratively build and test the initial SLU models until the UE expert has a set of self-consistent call types before creating the final annotation guide. The final annotation guide would then be used by the labelers to label all the utterance data needed to build the final SLU model. Thus, the SLU Toolset is critical for creating the call types defined in the annotation guide which in turn is needed to label the data for creating the final SLU.

Alternatively, the labeled utterances can easily be exported in a format compatible with the internally developed NLU Toolset if further SLU model tuning is to be performed by the NLU expert using just the command line interface.

3.5 Reporting

One of the reporting components is the Annotation Guide Generation Tool. The UE expert can use this at any time to automatically generate the annotation guide from the Processed Data. Other reporting components include summary statistics and spreadsheets containing utterance and call type information.

4 Results

The performance of the preprocessing techniques has been evaluated on several datasets from various industry sectors. Approximately 10,000 utterances were collected for each application and the results of the data reduction at each processing stage are shown in Table 1. The Redundancy R is given by

$$R = 1 - \frac{U}{N}$$

where U is the number of unique utterances after feature extraction and N is the number of original utterances.

Initial UE experts of the tools have been successful in producing annotation guides more quickly and with very good initial F-measures.

$$F = \frac{2 \bullet precision \bullet recall}{precision + recall}$$

They have also reported that the task is much less tedious and that they have done a better job of covering all of the significant utterance clusters. Further studies are required to generate quantitative measures of the performance of the toolset.

5 Future Work

In the future, the system could be improved using other representative utterance selection algorithms (e.g., selecting the utterance with the minimum string edit distance to all others).

The grammars for entity extraction were not tuned for these applications and it is expected that further data reduction will be obtained with improved grammars.

6 Conclusions

We presented an interactive speech data analysis system for creating and testing spoken language understanding systems. Spoken language understanding is a critical component of automated customer service applications. Creating effective SLU models is inherently a data driven process and requires considerable human intervention. The fact that this process relies heavily on human expertise prevents a total automation of the process. Our experience indicates that augmenting the human expertise with interactive data analysis techniques made possible by machine learning techniques can go a long way towards increasing the efficiency of the process and the quality of the final results. The automatic preprocessing of the utterance data prior to its use by the UE expert results in a considerable reduction in the number of utterances that needs to be manually examined. Clustering uncovers certain structures in the data that can then be refined by the UE expert. Supervised machine learning capabilities provided by interactive relevance feedback tend to capture the knowledge of the UE expert to create the guidelines for labeling the data. The ability to test the generated call types during the design process helps detect and remove problematic call types prior to their inclusion in the SLU model. This tool has been used to create the labeling guide for several applications by different UE experts. Aside from the increased efficiency and improved quality of the generated SLU systems, the tool has resulted in increased uniformity in the way different UE experts classify calls into call type labels.

Acknowledgements

We would like to thank Harris Drucker, Patrick Haffner, Steve Lewis, Maria Alvarez-Ryan, Barbara Hollister, Harry Blanchard, Liz Alba, Elliot Familant, Greg Pulz, David Neeves, Uyi Stewart, and Lan Zhang for their contributions to this work.

References

Harris Drucker, Behzad Shahraray, and David C. Gibbon, 2002. *Support Vector Machines: Relevance Feedback and Information Retrieval*, Information Processing and Management, 38(3):305-323.

Yaov Freund and Robert Schapire, 1999. *A Short Introduction to Boosting*, Journal of Japanese Society for Artificial Intelligence, 14(5):771-780.

Patrick Haffner, Gokhan Tur, and Jerry Wright, 2003. *Optimizing SVMs for complex Call Classification*, ICASSP 2003.

A. L. Gorin, G. Riccardi, and J. H. Wright. 1997. *How May I Help You?* Speech Communication, 23:113-127.

A. K. Jan and R. C. Dubes, 1988. *Algorithms for Clustering Data*, Prentice Hall.

M. F. Porter, 1980. *An Algorithm For Suffix Stripping*, Program, 14(3):130-137.

M. Rochery, R. Schapire, M. Rahim, N. Gupta, G. Riccardi, S. Bangalore, H. Alshawi and S. Douglas, 2002. *Combining prior knowledge and boosting for call classification in spoken language dialogue*, ICASSP 2002.

SAS Institute Press Release, 2002. *New SAS® Text Mining Software Surfaces Intelligence beyond the Numbers*, 1/21/02.

Robert Schapire and Yoram Singer, 2000. *BoosTexter: A Boosting-based System for Text Categorization*, Machine Learning, 39(2/3):135-168.

Gokhan Tur, Robert E. Schapire, and Dilek Hakkani-Tür, 2003. *Active Learning for Spoken Language Understanding*, Proceedings of International Conference on Acoustics, Speech and Signal Processing, ICASSP 2003.

V. N. Vapnik, 1998. *Statistical Learning Theory*, John Wiley & Sons, Inc.