A Novel Method for Speech Data Mining

Dai Sheng-Hui, Lin Gang-Yong, Zhou Hua-Qing
School of Information and Electronics, East China Institute of Technology, China, 344000

***Abstract*— Text-to-Speech (TTS) system is one to translate given text to speech which can be used in various applications such as information releasing systems, voice response devices, voice services in E-mail and reading machines for the blind. Great progress has been made in the research on Chinese TTS systems and several Chinese TTS systems have been published. However, because of the complexity of Chinese, the current available speech patterns are not very fine. The speech quality of those systems developed from these patterns is not good enough to meet the needs of users. The main purpose of this paper is to gain a refined prosodic model of Chinese speech. Traditional methods are not used in this thesis and data mining techniques are employed. Data mining is the process of discovering advantageous patterns in database. There are now many data mining algorithms, one of which is neural network. This paper presents a data mining system using clustering algorithm to find useful patterns from Chinese speech database. Study on the tone changes of Chinese two-word phrases has been made and good results have been achieved. They are helpful to develop high quality Chinese TTS systems.**

***Index Terms*—Classifiers, clustering, data reduction, relevance feedback, speech data mining**

I. INTRODUCTION

The field of speech data mining is in the midst of defining itself. As in previous debates on the nature of text data mining [22], we have multiple and sometimes overlapping areas. Does language modeling for information retrieval fall under the heading of speech data mining [23]? How about information extraction from speech [24] or speech summarization [25]? This is a rich area for study, and we wish to propose a slightly different tack that we feel is relevant to the field.

Our work on semantic data mining of short utterances relates to the design of a taxonomy that covers an initial set of utterances, with a specific set of utterance types. This taxonomy relates to a specific business problem of interest to the analyst, who is a subject matter expert in this specific business area. An effective taxonomy will be a set of utterance types such that this set of types covers the preponderance of the utterances in the utterance set. As an example, the utterance, "I want to order a calling card for my business line" would be mapped to the utterance type Request(Order_CallingCard). Utterances may have multiple types. The set of utterance types forms the taxonomy of interest, and each utterance type is a testable hypothesis when expressed as an NLU classifier. The overall goal is to develop an effective dialog response system for use in large-scale telephony applications. Initially, our research examined how relevance feedback might be used to augment active learning as part of the process of refining an NLU classifier that was deployed in the field and needed to adapt to a changing situation. Based on an initial investigation, we determined that the benefits of an interactive methodology with relevance feedback would yield minimal results at this stage of the process. However, we did find that this method could have significant impact in the initial creation phase of the set of NLU classifiers.

Relevance feedback is typically applied to full text documents; therefore, we did some initial experimentation to determine the value of this approach on short utterances [7]. We used over 12 000 utterances with 75 known utterance types from one of our existing applications and applied relevance feedback techniques to determine the coverage ratio. From this experiment, we determined the following.

- The coverage ratio was sufficient to warrant implementing the algorithm into an interactive system.
- Relevance feedback would not give good results on small sets (sets containing less than 1% of the total number of utterances).

Before we can create an effective dialogue response system for a particular application, we collect thousands of utterances in order to effectively cover the space. Initial data collection is done through a "wizard" system that collects the set of utterances in the context of the specific business problem [11]. Once collected, these utterances are transcribed by hand and turned over to the analyst who classifies the utterances and develops a labeling guide that documents the taxonomy. This taxonomy forms the basis for a set of Natural Language Understanding (NLU) classifiers, which have a one-to-one relationship with the set of utterance types. At this point, a separate group of people, called labelers, use the labeling guide as the basis to classify a larger set of utterances. Once the utterances are classified, they serve as input to build the NLU classifiers. The ultimate goal would be an effective set of NLU classifiers that could be used with a dialogue manager that will understand and properly reply to people calling in to a telephone voice response unit [10].

We test the NLU classifiers in the field to determine their effectiveness in combination with the dialog manager. In many instances, this combination may not completely satisfy the business problem. This initiates an interactive process that often requires an adjustment to

Manuscript received Oct 15, 2009; revised Nov 15, 2009; accepted Dec 25, 2009.

doi:10.4304/jsw.6.1.64-71

the taxonomy. As we worked with the analysts to refine this interactive process, we adapted our methodology to incorporate their feedback and comments. We determined that many of the utterances were either exact duplicates or so similar that the NLU classifier would recognize them as duplicates. We decided to incorporate data reduction methods to identify these "clones" and hide them from the analyst while still making them available to the NLU creation phase. Other feedback from the analysts indicated that they wanted methods for seeding relevance feedback iterations that went beyond simple search. We determined that clustering the utterances could give approximations to the utterance types that the analyst could then iteratively improve. Our goal in creating these interactive techniques is to save time for the analyst and help generate more consistent results when a project is handed off from one analyst to another.

In this paper, we will show how and why we adapted the following techniques to work on short utterances:

• data reduction;
• clustering;
• relevance feedback.

In addition, we produce an NLU metric that gives a measure of accuracy for the coverage of the taxonomy. Using this metric, an analyst can refine the taxonomy before it goes to the labelers and especially before it goes to the field.

II. Data Reduction

After data collection, the utterances or documents are mapped into a feature vector space for subsequent processing. In many applications, this is a one-to-one mapping, but in cases where the documents are very short (e.g., single sentences or phrases), this mapping is naturally many-to-one. This is obviously true for repeated documents, but in many applications, it is desirable to expand the mapping such that families of similar documents are mapped to a single feature vector representation. For many speech data collections, utterance redundancy (and even repetition) is inherent in the collection process, and this is tedious for analysts to deal with as they examine and work with the dataset. Natural language processing techniques including text normalization, called entity extraction, and feature computation are used to coalesce similar documents and thereby reduce 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). Analysts may choose to expand these clone families to view individual members, but the bulk of the interaction only involves a single representative utterance from each set.

A. Text Normalization

In data reduction, we must carefully define what is meant when we say that utterances are "similar." There is no doubt that the user interface does not need to display 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" versus "one hundred" or "$50" versus "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 versus I would" or "I want to" versus "I want to." Acronym expansions can be included here: "I forgot my personal identification number" versus "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 an NLU classifier). We could go further and include synonyms or synonymous phrases: "I want" versus "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.") In addition, 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).

B. Named Entity Extraction

Utterances that differ only by an entity value should also be collapsed. For example, "give me extension 12 345" 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 items 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.

C. 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 [21], weighting features based on the number of times a word appears in an utterance (Term

Frequency; TF) or how unusual the word is in the corpus (Term frequency—inverse document frequency; TF-IDF) [20], and performing word stemming [12]. 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 utterance types, it is appropriate to use a different set of features when training NLU classifiers with large amounts of data. 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.

TABLE I TYPICAL REDUNDANCY RATES FOR COLLECTIONS OF CUSTOMER CARE DATA

Industry Sector	Financial	Health Care	Insurance	Retail
Original Utterances	11,623	12.080	12,109	10,240
Unique Utterances	10,021	10,255	8,865	4956
Unique Utterances after Text Normalization	9,670	9,452	8,103	4,392
Unique Utterances after Entity Extraction	9,165	9,382	7,963	4,318
Unique Utterances after Feature Extraction	7,929	7,946	6,530	3,566
Redundancy	31.8%	34.2%	46.1%	65.2%

D. Data-Reduction Results

The effectiveness of the redundancy removal is largely determined by the nature of the data. As shown in Table I, we have found typical redundancy rates for collections of customer care data of from 30 to 40%. In some cases, where the task is less complex, we have observed data redundancy greater than 50%. Note that as the average length of the documents increases, the redundancy decreases.

III. CLUSTERING

While removing redundant data greatly eases the burden on the analyst, we can go a step further by organizing the data into clusters of similar utterances. Unfortunately, available distance metrics for utterance similarity are feature-based and result in lexical clusters rather than clusters of semantically similar utterances.

Therefore, the goal of this stage of the processing is to add further structure to the collected utterance set so that an analyst can more easily make informed judgments to define the utterance types. Clustering short utterances is problematic due to the paucity of available lexical features. It is quite common for two utterances to have no common features; this is not the case when clustering long-form documents such as news stories.

In this section, we address this issue and present an efficient method for clustering utterance data.

A. Clustering Algorithm

Clustering causes data to be grouped based on intrinsic similarities. After the data-reduction steps described above, clustering serves as a bootstrapping process for creating an initial reasonable set of utterance types. In any clustering algorithm, we need to define the similarity

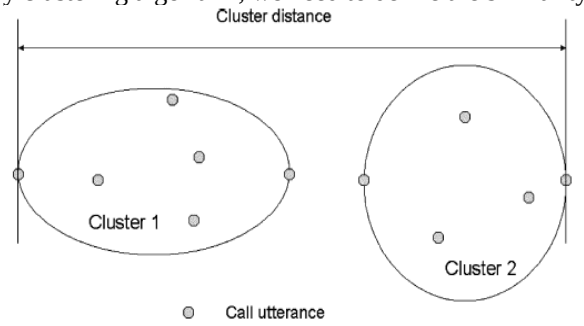

Fig. 1. Illustration of cluster distance.

(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 short utterances of words. Each utterance is converted into a feature vector, which is an array of terms (words) and their weights. The distance between two utterances is defined as the cosine distance between corresponding feature vectors. Assume that x and y are two feature vectors and that the distance d(x, y) between them is given by

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

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-based weights. 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 each utterance to generate the set of feature vectors.

The distance between two clusters is defined as the maximum utterance distance between all pairs of utterances: one from each cluster. Fig. 1 illustrates the definition of the cluster distance. The range of utterance distance values is normalized from 0 to 1, as is the range of the cluster distance values. When the cluster distance is 1, it means that there exists at least one pair of utterances–one from each cluster–that is totally different (sharing no common term). Many clustering algorithms can be found in an excellent reviewing paper by Jain et al. [5]. Buhman et al. [17] proposed maximum entropy approach for pair wise data clustering. However, it does not generate the clustering tree, which is required in our system to efficiently recreate clusters by cutting the dendrogram at different levels. Guha et al. [18] introduced the Hierarchical Agglomerative Clustering (HAC) algorithm called Clustering Using Representatives (CURE). CURE represents a cluster by a fixed number of points scattered around it, which makes the algorithm insensitive to the outliers and more efficient for large data sets. Karypis et al. [19] proposed the Chameleon

algorithm, which is a hierarchical clustering using dynamic modeling. A key feature of the Chameleon algorithm is that it accounts for both inter connectivity and closeness in identifying the most similar pair of clusters. Both CURE and Chameleon are efficient, but the clustering results are normally the approximations of those of traditional HAC. Our interest is in a clustering method that generates the same results as traditional HAC, yet in an efficient way. In the following, we will first briefly describe the traditional HAC and then show how we improve it effectively reduce both the time and the space complexities in our specific application. The details of the traditional hierarchical agglomerative clustering algorithm can be found in [4]. 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 all utterances are in one cluster such that a full dendrogram is created. We then cut the clustering tree using a preset threshold to generate the clustering results. Depending on the analyst's requirements for clustering, the optimal threshold will vary. We allow for the option of generating new clusters using a different threshold.

The principle of HAC is straightforward, yet the computational complexity and memory requirements are high for large datasets. Assuming that there are utterances, direct implementation of HAC requires $O(N^2)$ memory for storing the utterance distance matrix and cluster distance matrix. Given that the average size of the utterances is small (10 terms) compared to the feature dimension (10 k), there is an efficient way to compute the distance between two utterances. From formula (1), we know that the norm of each utterance ||x|| is 1.0 after feature normalization, and x*y can be computed by checking only the nonzero terms for both utterances. Therefore, instead of maintaining the huge utterance/cluster distance matrix, we compute the utterance/cluster distance on the fly, such that the memory usage is effectively reduced to O(N).

Another interesting phenomena is that when the utterances are short, a significant number of entries in the utterance distance matrix are 1.0 since x*y=0 if x and y share no common terms. This also means that in the clustering procedure, for each cluster, most of the distances from other clusters are 1.0. Since the distance from one cluster to its nearest neighbor never decreases, once it is 1.0, these clusters need not be considered for merging in future iterations. To further improve the speed, instead of searching the nearest clusters among all pairs of clusters $O(N^2)$, for each cluster, we keep track of its neighboring clusters and corresponding distances, where $k<<N$, such that we only need to search O(N) distance to locate the closest clusters. The overhead is the maintenance of the neighboring clusters for all clusters. When two clusters merge, we only need to update those clusters whose neighbors contain at least one of the merged clusters. Therefore, the maintenance is minimal.

Table II shows the computation time and memory usage for directHAC implementation and our improved version. We compared them on two datasets: one contains 5000 utterances and the other 20 000 utterances. For the first dataset, the direct HAC implementation requires 4 hr to complete and uses 200-MB memory, yet the improved implementation only takes 15 s and requires 8-MB memory. For the second dataset, we only provide the results for the improved implementation and the memory usage for the direct implementation. We did not measure the computation time since it takes too long—a reasonable estimate is about 250 hr.

TABLE II CLUSTERING ALGORITHM COMPLEXITY

Implementation	Number of Utterances			
	5,000		20,000	
	Computation Time	Memory (MB)	Computation time	memory (MB)
Direct HAC	4 Hour	200	N/A	~3200
Improved HAC	15 seconds	8	540 seconds	30

B. Merging Clusters

As mentioned before, HAC may still produce a large number of clusters since the utterances are short. To reduce the total number of clusters, we merge all clusters smaller than an established minimum into a special "other" cluster. While there is no set rule for the minimum size of clusters, we find that a minimum of three to five are reasonable choices in our study.

Anecdotally, the analysts found it easier to transform a set of clusters into utterance types than to create utterance types directly from a large set of flat data. The specific utterance types depend on the business problem that the analyst is attempting to solve. 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 utterance type may be scattered into several clusters, and the analyst has to merge these clusters to create the desired utterance type. On the other hand, if the clustering is aggressive, there may be multiple utterance types in one cluster, and the analyst needs to manually split the mixture cluster into different utterance types. In real applications, we tend to set a relatively low threshold since it is easier to merge small homogeneous clusters into a larger cluster than it is to split one big heterogeneous cluster into many smaller clusters.

C. Clustering Performance Evaluation

We use the purity concept explained in [6] to evaluate clustering performance. The two measurements are the average cluster purity (ACP) and the average utterance type purity (ATP), as explained below. First, we define the following:

n_{ij} total number of utterances in cluster i with utterance type j;
N_T total number of utterance types;
N_C total number of clusters;
N total number of utterances;
$n_{.j}$ total number of utterances with utterance type j;
$n_{i.}$ total number of utterances in cluster i.

The purity of a cluster $p_{i.}$ can then be defined as

$$p_{i.} = \sum_{j=1}^{N_T} \frac{n^2_{ij}}{n^2_{i}}$$

and the average cluster purity (ACP) is

$$ACP = \frac{1}{N} \sum_{i=1}^{N_C} p_{i.} \bullet n_{i.}$$

Similarly, the utterance type purity and the average utterance type purity (ATP) are calculated as

$$p_{.j} = \sum_{i=1}^{N_C} \frac{n^2_{ij}}{n^2_{.j}}$$

$$ATP = \frac{1}{N} \sum_{j=1}^{N_T} p_{.j} \bullet n_{.j}$$

The ATP measures how well the utterances of one utterance type are limited to only one cluster, and the ACP measures how well the utterances in one cluster are within the same utterance type. Two extreme cases are 1) if all utterances are in one cluster, then ATP=100%, and ACP is small; 2) if each utterance is in a separate cluster, then ACP =100%, and ATP is small. Ideally, we prefer a high ACP and a high ATP for each cluster. When this is not the case (given that the clustering algorithm is used for bootstrapping the utterance types), we prefer a high ACP with reasonable ATP over a high ATP with low ACP (see Table III). In this mode, the analyst does not need to spend too much effort on checking the consistency of each cluster but rather study the difference and similarity among clusters.

D. Clustering Results

We evaluated the clustering performance on four applications across four different industrial sectors, specifically, financial, health care, insurance, and retail. To cope with the multiple labels problem, we only consider the single label utterances in the evaluation. Table III gives the results. In the evaluation, we set the clustering distance threshold to 0.6, and the minimum cluster size to 5.

From Table III, we see that the ACP is in the same range, roughly 60–70% across the four applications, and the ATP are quite different among the different applications. The health care application achieves the highest ATP with 45.6%, and the financial application achieves the lowest ATP, with 23.2%. Generally, when the number of utterance types is larger, the ACP will be smaller, and when the number of clusters is larger, the ATP will be smaller. For the financial and retail applications, the numbers of utterance types are small; therefore, their corresponding ACPs are large. For the health care and insurance applications, the numbers of utterance types are large, and therefore, their ACPs are small. For financial and insurance applications, the numbers of clusters are large, and their ATPs are small. The health care and retail applications have a small number of clusters, and their ATPs are large.

The utterance types for different applications are determined by analysts based on their knowledge and on the business problem to be solved. Therefore, the number of utterance types may not uniformly reflect the scattering in the datasets. The clusters are determined in a systematic way, and they more reliably indicate the syntactic structure of the datasets. For example, the financial application has 36 utterance types, but it does not mean that the dataset is homogeneous. Actually, it is not homogeneous since there are a large number of clusters, which implies that the dataset is actually heterogeneous.

TABLE III CLUSTERING PERFORMANCE RESULTS BY APPLICATION

Data	N_C	N_T	ATP (%)	ACP(%)
Financial	36	335	23.2	71.2
Health care	92	133	45.6	61.3
Insurance	51	279	25.4	60.2
Retail	31	131	35.8	70.2

IV. RELEVANCE FEEDBACK

Although clustering provides a good starting point, finding all representative utterances belonging to one utterance type is not a trivial task. Additional data-mining tools are desirable to help the analyst. Our solution is to provide a classification mechanism based on [14] SVM classifiers for the analyst to perform this tedious task. In such classification-based approaches, the user sequentially assigns labels to examples until the examples belonging to the target utterance type are reasonably separated from the rest.

A. Support Vector Machines

We adopted SVMs as the classifier [7], [13] for several reasons: First, SVMs efficiently handle high dimensional data. In our case, this is a set of utterances with a large vocabulary. We compared SVMs to Adaboost [2], which is another classification algorithm used for creating NLU models. On three customer care data sets, corresponding to medical and telecommunication applications and containing between 9000 and 35 000 examples, the F-measures obtained with Adaboost ranged from 67 to 77. The use of SVMs improved these F-measures by three points on average.

Second, SVMs provide especially reliable performance with a small amount of training data. If we reduce the size of the aforementioned data sets to less than 1000 examples, SVM F-measure is, on average, five points over Adaboost F-measure.

Third, with the use of Rational Kernels [16], SVMs can be extended to accept as input the word lattice produced by an ASR system instead of just the best recognized sentence. A characteristic of SVMs is that all the necessary computations only require inner products between vectors belonging to the input space. Taking two vectors x and y, their inner product (x.y) can be replaced by the application of a kernel function K(x,y). Provided that it verifies some properties (symmetric, positive definite), this kernel can engender an inner product (and,

consequently, a metric) in a feature space that is different from the input space. Now, suppose that instead of the feature extracted from sentences as described in Section II, the output of the speech recognizer consists of word lattices, which are encoded as graphs or finite state machines (FSMs). While there is no obvious inner product in the space of FSMs, the application of kernels (with a called Rational) to these FSMs defines a metric that can be used to apply SVM learning to these FSMs.

In our ASR system, the oracle accuracy, which is defined as the accuracy of the path in the lattice closest to the transcription, is considerably better than the word accuracy using best recognized sentence only. This means that often, the correct sentence is somewhere in the lattice but not as the top candidate. Rational Kernels, by using the additional information contained in the lattice, improve the F-measures [9] observed in the three data sets by 1.5 points on average. (F-measure combines recall (r) and precision (p) with equal weight as follows: $F(r,p)=2rp/(r+p)$).

Rational Kernels have also been used to generalize the clustering algorithm described in Section III-A to FSMs. Their actual use in an environment that requires user interaction is under investigation (with the challenge of visualizing FSMs).

B. Relevance Feedback

The most commonly used approach in Information Retrieval (IR) is relevance feedback, which is a form of query-free retrieval where documents are retrieved from a collection according to a measure of relevance to a given set of documents. In essence, an analyst indicates to the retrieval system that it should retrieve "more documents like the ones desired and not like the ones ignored." Selecting relevant documents based on analyst's inputs is basically a classification (relevant/irrelevant) problem. Relevance feedback is an iterative procedure. The analyst starts with a cluster or a query result by certain keywords and marks each utterance as either a positive or negative utterance for the utterance type. The analyst'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 utterance 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 analyst, and the second loop of relevance feedback begins. The analyst determines the end of this cycle based on how closely the utterances in the generated set match their concept for the utterance type.

For efficient labeling of large quantities of data, another iterative approach, generally referred to as active learning, is preferred. The most relevant utterances, while interesting from an IR standpoint, are usually obvious for the classifier. They are not those that maximize progress when learning them. It is rather the labeling of uncertain utterances, which lie at the decision boundary, which gives the greatest improvement to the discrimination between relevant and irrelevant utterances. To establish which utterances lay at the decision boundary, one can rely on either geometric or probabilistic criteria.

According to the geometric criterion, the examples that should be labeled in priority stand at the center of the classifier. For an example x, let g(x) be the output of the SVM before the addition of any bias. The geometric criterion relies on the transformation

$$g(x)+b$$

such that for positive support vectors

$$g(x)+b=1$$

and for negative support vectors

$$g(x)+b=-1$$

The center of the margin corresponds to

$$g(x)+b=0$$

In our problem, we define the positive class as examples belonging to the utterance type and the negative class as all other examples. As a consequence, the positive class has many fewer representatives than the negative class. Therefore, choosing examples at the center of the margin will typically return a large majority of negative utterances and result in a labeling process that is both suboptimal and frustrating.

The probabilistic criterion relies on the fact that the classifier output approximates in a reasonable way the posterior probability that a given utterance belongs to the utterance type and selects examples where the posterior probability is the closest to 0.5. In the case of SVMs, such a probabilistic approximation can be obtained with the application of univariate logistic regression to the output of the SVM [14]. The transformation consists of

$$g'(x)=\sigma(a.g(x)+b)$$

where σ is the sigmoid function, and a and b are optimized to minimize the Kullback–Leibler divergence [14] between g'(x) and the posterior probability of the class $P(c|x)$, given x. Separate training sets should be used to train the SVM classifier and the logistic parameters a and b. We use cross-validation to maximize the use of labeled examples. Note that in the case of active learning, our logistic remapping function is trained on the already-labeled examples, whose distribution is skewed and not statistically representative of the true distribution. Despite this limitation, we found that the logistic remapping approximation worked well on unlabeled examples, returning comparable numbers of positive and negative examples, and converging significantly faster than the geometric criterion. Both theory and computer simulations predict that active learning, using the probabilistic criterion, minimize the number of examples one has to label to achieve a given classification accuracy on test data. Our simulations suggest that if the goal is to label enough examples to build a classifier that generalize well on test data, the active learning strategy can reduce by up to a factor of six the number of examples that need to be labeled.

However, the reality is quite different. First, the initial goal is not to build a classifier but rather to collect typical examples and estimate the coverage of an utterance type for the design of a labeling guide. Second, active learning typically returns the hardest to classify examples. Each one of them has to be examined carefully by the analyst who needs to have a very good idea of what kind of utterances a given utterance type covers. Relevance

feedback returns a lot of "obvious," and sometimes nearly identical, examples. This can be frustrating but has the following advantages over active learning:

• In many cases, all the utterances are obvious positives, and a single "select all" click will do the job. Thus, labeling ten examples using relevance feedback can be as fast as labeling one example using active learning.

• To return the most relevant examples gives the analyst clear feedback that he/she is going in the right direction. Boundary examples returned by active learning do not give a clear indication of how well the classifier works on previously labeled examples.

• During the initial iterations, the analyst may not have a clear idea of the full coverage of an utterance type and needs to be confronted with data to better specify the utterance type. By presenting "typical" examples, relevance feedback works better in this loosely defined search scenario.

Moreover, some of these "typical" examples can be set apart as illustrations for the labeling guide. In practice, all examples whose posterior probability of belonging to the utterance type ranges from 0.5 to 1 are returned and sorted in descending order of probability. The analyst can then choose to label this list either from the top or the bottom. The general preference seems to go toward labeling the most relevant (top) examples, at least at the beginning of the labeling process.

V. NLU METRIC

The analyst can improve utterance types by iteratively building and testing interim NLU classifiers. A Web interface was added to allow the analyst to build and test NLU classifiers and to better understand patterns in the NLU classifier test results. We used BoosTexter as the underlying boosting algorithm for classification [1]–[3].

After the analyst has labeled the utterances (we will refer to these as truth utterance-type labels), approximately 20% of the labeled utterances are set aside for testing. The remaining data are used to build the initial NLU classifier. For each of the tested utterances in the test data, logs show the classification confidence scores for each utterance type. Confidence scores are replaced by probabilities that have been computed using a logistic function. These probabilities are then used to calculate the NLU metric, which attempts to reveal patterns in the classification results. The NLU metric, roughly speaking, is a measure of utterance type differentiability. The NLU metric is calculated as follows and is averaged over the utterances that belong to only one utterance type:

$$S=\begin{cases}\frac{1}{N}\sum_{i=1}^{N}(T_i - X_i), \text{ for } T_i = H_i (\textit{correctly classified})\\ \frac{1}{N}\sum_{i=1}^{N}(T_i - H_i), \text{ for } T_i = H_i (\textit{correctly classified})\end{cases}$$

Where S is the NLU Metric, N is the number of utterances that belong to only one utterance type, T_i is the truth probability, X_i is the next highest probability, and H_i is the highest probability. A test utterance is correctly classified if the calculated probability of the truth type is the highest probability.

Table IV shows two sample test utterances and the test log results. The first utterance is incorrectly classified (shown in italics) since the Request(Sales) utterance type has a higher probability than the Request(Order_CC) utterance type. In this particular case, the word "order" also figures prominently in the Request(Sales) utterance type. As can be expected, this overlapping language problem will occur at times, no matter how much work is expended to create distinct utterance types. The second utterance is correctly classified, but the probabilities are too close. Ideally, the truth utterance type probability is near 1, and the next nearest probability is close to 0 so that the contribution to the NLU metric would be close to 1. If the probabilities are too close together, then these two utterance types can be confused in the field, and calls can possibly be incorrectly routed. In this case, the contributions to the NLU metric from these two utterances were -0.5091 (incorrectly classified) and 0.0495 (correctly classified). As can be seen in Table V, the NLU metric for the Request (Order_CC) utterance type is 0.681. Of the 18 test utterances, only two were incorrectly classified. Thus, although some of the test utterances in the test log indicate problems, on the aggregate, the NLU metric for this utterance type is quite good. Other good utterance types are shown in Table V. If the NLU metric was less than 0.50 or negative, this would indicate a problem with the utterance type. The best approach for the analyst is to evaluate both the test log probabilities (for utterance level problems) and the NLU metric (for aggregate level problems) for every utterance type. This metric allows the analyst to identify utterance types that might have problems in the field. Once identified, the analyst could redefine the problematic utterance types. Another interim NLU classifier could then be built and tested to determine if the changes improved the utterance type. The analyst can iteratively build and test the interim NLU classifiers. Once the utterance types are correct, the final annotation guide is created. The final annotation guide would then be used by the labelers to label all the utterance data needed to build the final NLU classifier. The NLU metric helps create better utterance types, which ultimately leads to a better NLU classifier.

TABLE IV TEST LOG PROBABILITIES

Utterance	Truth Utterance type [Probability]	Other Utterance types [Probability]
order a calling	*Request(Order_CC) [0.4571]*	***Request(Sales) [0.9662]***
i wanna order a calling card for my business line	**Request(Order_CC) [1.0000]**	Request(Status_Order) [0.9505]

TABLE V NLU METRIC

Utterance type	# of Tests (# correct)	NLU Metric
Report(LostStolen_CC)	31 (30 correct)	0.941
Request(Call_Transfer_CSR)	28 (27 correct)	0.850
Request(Order_CC)	18 (16 correct)	0.681

VI. SUMMARY

We have shown adaptations of text-based data mining tools to make them more useful in the context of speech data mining. These tools enable our analysts to develop NLU classifiers in the context of a specific business problem. Data-reduction techniques are used to take advantage of the nature of short utterances and give a compacted view of a large data set. In clustering, we discussed the difficulties of creating clusters from short utterances. We also discussed how we took advantage of the distance metric for short utterances to help us improve the performance of the clustering algorithm. For relevance feedback on short utterances, we have shown that using SVMs and then reporting results that are sorted by distance from the support vector gives results that are more useful for our analysts. Our analysts have reported that the task is much less tedious and that they have done a better job of covering all of the significant utterance types. The NLU metric that we created gives us a method of determining the accuracy of the NLU before it goes into the field.

REFERENCES

[1] Y. Freund and R. Schapire, "A short introduction to boosting," *J. Japanese Soc. Artificial Intell.*, vol. 14, no. 5, pp. 771–780,2008.
[2] M. Rochery, R. Schapire, M. Rahim, N. Gupta, G. Riccardi, S. Bangalore, H. Alshawi, and S. Douglas, "Combining prior knowledge and boosting for call classification in spoken language dialogue," in *Proc. ICASSP*.
[3] R. Schapire and Y. Singer, "BoosTexter: a boosting-based system for text categorization," *Machine Learning*, vol. 39, no. 2/3, pp. 135–168, 2008.
[4] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*. Englewood Cliffs, NJ: Prentice-Hall,2008.
[5] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," *ACM Comput. Surveys*, vol. 31, no. 3, pp. 264–323, Sep.2008.
[6] J. Ajmera, H. Bourlard, I. Lapidot, and I. McCowan, "Unknown-multiple speaker clustering using HMM," in *Proc. ICSLP*, Denver, CO, 2008, pp. 573–576.
[7] I. Drucker, I. Gibbon, and I. Shahraray, Support vector machines: relevance feedback and information retrieval, in Inf. Process. Manage., vol. 28, pp. 305–332, 2002.
[8] H. Drucker, D. Gibbon, and B. Shahraray, "Relevance feedback using support vector machines," in *Proc. Int. Conf. Machine Learning*.
[9] C. Van Rijsbergen, *Information Retrieval*, Second ed. London, U.K.: Butterworth, 1979.
[10] A. Abella and A. Gorin, "Construct algebra: analytical dialog management," in *Proc. Annu. Meet. Assoc. Computat. Linguistics*, Washington, DC, Jun. 1999.
[11] A. L. Gorin, G. Riccardi, and J. H. Wright, "How may I help you?," in *Speech Commun.*, vol. 23, 1997, pp. 113–127.
[12] M. F. Porter, "An algorithm for suffix stripping," *Program*, vol. 14, no. 3, pp. 130–137, 1980.
[13] V. N. Vapnick, *Statistical Learning Theory*. New York: Wiley, 1998.
[14] C. Platt, "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods," in *Advances in Large Margin Classifiers*, A. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, Eds. Cambridge, MA: MIT Press, 1999, pp. 61–74.
[15] Z. Xu, X. Xu, K. Yu, V. Tresp, and J. Wang, "A hybrid relevance-feedback approach to text retrieval," in *Proc. 25th Eur. Conf. Inf. Retrieval Res.*, Pisa, Italy, Apr. 14–16, 2008.
[16] C. Cortes, P. Haffner, and M. Mohri, "Rational Kernels: theory and algorithms," *J. Machine Learning Res.*, vol. 5, pp. 1035–1062, August 2004.
[17] J. M. Buhman and T. Hofmann, "A maximum entropy approach to pairwise data clustering," in *Proc. Int. Conf. Pattern Recogn.*, 1994, pp. 207–212.
[18] S. Guha, R. Rastogi, and K. Shim, *CURE: An Efficient Clustering Algorithm for Large Databases.* Seattle, WA: SIGMOD, Jun. 1998, pp. 73–84.
[19] G. Karypis, E.-H. Han, and V. Kumar, "CHAMELEON: hierarchical clustering using dynamic modeling," *IEEE Comput.*, vol. 32, no. 8, pp. 68–75, Aug. 1999.
[20] G. Salton and C. Buckley, "Term weighting approaches in automatic text retrieval," *Inf. Process. Manage.*, vol. 5, no. 24, pp. 513–523, 1988.
[21] J. Chu-Carroll and B. Carpenter, Dialogue management in vector-based call routing, in Comput. Linguistics, 1998.
[22] M. A. Hearst, "Untangling text data mining," in *Proc. ACL 37th Annu. Meet. Assoc. Comput. Linguistics*. College Park, MD, Jun. 20–26, 1999.
[23] J. Lafferty and C. Zhai, "Document language models, query models, and risk minimization for information retrieval," in *Proc. ACM SIGIR Conf. Res. Development Inf. Retrieval*, 2008.
[24] M. Jansche and S. Abney, "Information extraction from voicemail transcripts," in *Proc. EMNL*, 2002.
[25] A. Inoue, T. Mikami, and Y. Yamashita, "Improvement of speech summarization using prosodic information," in *Proc. ICASSP*, 2004, pp. 599–602.

Sheng-Hui Dai received the B.S. and M.S. degrees in computer science from East China Institute of Technology, Fuzhou, China, in 1999, 2006, respectively. He was an assistant of Information Engineering at East China Institute of Technology, Fuzhou, China, in 1999, and became a lecturer in 2004. His research interests are in data mining, real-time systems, multimedia, mobile computing, and wireless communications.