OPTIMIZATION OF TEXT DATABASE USING HIERACHICAL CLUSTERING

Jilei Tian[1] *and Jani Nurminen*[2]

[1]Media Laboratory, Nokia Research Center, Tampere, Finland
[2]Nokia, Devices R&D, Tampere, Finland
{jilei.tian, jani.k.nurminen}@nokia.com

ABSTRACT

Many speech and language related techniques employ models that are trained using text data. In this paper, we introduce a novel method for selecting optimized training sets from text databases. The coverage of the subset selected for training is optimized using hierarchical clustering and the generalized Levenshtein distance. The validity of the proposed subset optimization technique is verified in a data-driven syllabification task. The results clearly indicate that the proposed approach meaningfully optimizes the training set, which in turn improves the quality of the trained model. Compared to the existing state-of-the-art data selection technique, the proposed hierarchical clustering approach improves the compactness of data clusters, decreases the computational complexity and makes data set selection scalable. The presented idea can be used in a wide variety of language processing applications that require training with text data.

Index Terms— hierarchical clustering, Levenshten distance, text data selection

1 INTRODUCTION

Most of the current speech and language processing systems contain models that have to be trained with text data. For example, in pronunciation modeling, data-driven approaches such as neural network based or decision tree based methods [4], are often applied, especially for unstructured languages such as English. These statistical models are trained using a pronunciation dictionary containing grapheme-to-phoneme entries. In text-based language identification [6], the model is trained using a multilingual text corpus that consists of word entries from the target languages. In the data-driven syllabification task [5], the model is trained using text-based pronunciations and the corresponding syllable structures.

The text databases used for the training are often built off-line by collecting all the possible entries from the available language resources. The selected data is then used as a training corpus for data-driven modeling, or for extracting rules. For improving the quality of database, it is common to carry out manual corrections that require language-specific skills and that are time-consuming and expensive and in practice rather error-prone.

The use of large amounts of training data for model training often leads to high memory footprint and computational complexity. Because it is likely that the data contains unnecessary redundancy, it is rational to remove entries that are too similar to some other entry, while still achieving a sufficient coverage on the text entries and the contexts. Such training set optimization can result in enhanced model performance and naturally also offers the possibility to use a smaller training set size without sacrificing the performance. In practice, the reduced training set size brings two significant additional benefits. First, the amount of manual annotation work is reduced, which in turn decreases the probability of errors and inconsistencies in the annotations. Second, the memory consumption and the computational load caused by the training process are lowered.

Despite the potential benefits, the topic of training set selection is, however, often neglected in practice. If a reduced data set size is used, the subset is usually obtained by collecting a set of random entries from a larger text database or by decimating a sorted corpus. These simple methods do not optimize the coverage in any way and thus do not guarantee good performance. In [7], an objective function was defined and optimized to find the data that has the maximum distance and subsequently good coverage within the selected set. The technique was found to be effective but this solution has very high computational complexity.

In this paper, we present a method that can efficiently generate a subset from a text database in such a manner that the text coverage is maximized. To achieve this, the hierarchical cluster tree technique is applied in the subset optimization. The distance is measured using the generalized Levenshtein distances between the text strings. The proposed algorithm has a low computational complexity compared to the maximized objective function method. To demonstrate the usefulness of the proposed approach, we evaluate it in the data-driven syllabification task. Even though we chose this task, it should also be noted that it is fairly easy to the proposed method in a wide variety of different applications. In essence, the proposed approach could also be used for quantizating text data.

The remaining parts of this paper are organized as follows. First, Section 2 describes the distance measure and basic principles of the text database optimization algorithms. The data-driven syllabification task used in the experiments is introduced in Section 3. In Section 4, the performance of the proposed approaches is evaluated. Finally, some concluding remarks are presented in Section 5.

2 TEXT DATA OPTIMIZATION

2.1 Generalized Levenshtein distance

The generalized Levenshtein distance (GLD) is defined as the minimum cost of transforming one string into another by means of a sequence of basic transformations: insertion, deletion and substitution [3]. The transformation cost is determined by the costs assigned to each basic transformation.

Let *x* and *y* denote strings of length *m* and *n*, respectively, whose symbols belong to a finite alphabet of size *s*. Also, let x_i be the *i*th symbol of string *x*, with $1 \le i \le m$, and $x(i)$ denotes the prefix of the string *x* of length *i*, i.e. the substring containing the first *i* symbols of *x*. In addition, let $d(i,j)$ be the distance between $x(i)$ and $y(j)$, and ε denotes an empty string. Furthermore, we denote by $w(a,b)$, $w(a,\varepsilon)$ and $w(\varepsilon,b)$ the cost of substituting the symbol *a* with the symbol *b*, the cost of deleting *a* and the cost of inserting *b*, respectively. The distance $d(m,n)$ is recursively computed based on the definitions of $d(0,0)$, $d(i,0)$ and $d(0,j)$ ($i = 1\ldots m$, $j = 1\ldots n$), representing the initial distance, the cost of deleting the prefix $x(i)$ and the cost of inserting the prefix $y(j)$, respectively, as follows:

$$\begin{aligned} &d(0,0) = 0 \\ &d(i,0) = d(i-1,0) + w(x_i,\varepsilon) \qquad \forall i = 1\ldots,m \\ &d(0,j) = d(0,j-1) + w(\varepsilon,y_j) \qquad \forall j = 1\ldots,n \end{aligned} \tag{1}$$

$$d(i,j) = \min \begin{cases} d(i-1,j) + w(x_i,\varepsilon) \\ d(i,j-1) + w(\varepsilon,y_j) \\ d(i-1,j-1) + w(x_i,y_j) \end{cases} \tag{2}$$

The original Levenshtein distance is characterized by the following costs: $w(a,\varepsilon) = 1$, $w(\varepsilon,b) = 1$, and $w(a,b)$ is 0 if *a* is equal to *b* and 1 otherwise. Its generalized version assumes that variable costs can be associated to the different transformations involving the symbols.

2.2 Objective function maximization

In the previously proposed approach [7], an optimized text subset is generated by maximizing an objective function. As described in Section 2.1, the Levenshtein distance can be used for measuring the distance between any pair of entries. Similarly, the distance for the whole text data set can be calculated by averaging the distances of all the string pairs in the set. Suppose that there are *m* entries in the database and the *i*th entry is denoted by $e(i)$. With these definitions, we can compute the overall subset distance *D* as:

$$D = \frac{2 \cdot \sum_{i=1}^{m} \sum_{j>i}^{m} ld(e(i),e(j))}{m \cdot (m-1)}, \tag{3}$$

where $ld(e(i), e(j))$ is the GLD between the *i*th and *j*th entries.

The algorithm is proposed to construct the subset by recursively selecting the new entry that maximizes the subset distance. Assuming that the selected subset has *k* entries, the target is to find the *k*+1-th entry to the subset. The selection that approximately maximizes the amount of new information brought into the subset can be done as follows:

$$p = \underset{(1 \le i \le m)}{\operatorname{argmax}} \left\{ \sum_{j=1, e(i) \ne subset\_e(j)}^{k} ld(e(i), subset\_e(j)) \right\}. \tag{4}$$

A new selected entry *p* is repeatedly added into the subset as the *k*+1-th entry until the predefined subset size is reached.

2.3 Hierarchical cluster tree

In this paper, an efficient and scalable clustering based scheme is proposed for text database optimization. Since the text entries are composed of symbolic strings, it is not feasible to apply the conventional clustering methods such as the well-known K-means algorithm. However, the distance D_{ij} between entry *i*th and *j*th pair can be measured using Levenshtein distance as shown in Equation (1-2). Given the distances among all possible entry pairs of the given text database, the hierarchical clustering algorithm can be applied directly on the distance arrays.

Hierarchical clustering offers a way to investigate grouping data, simultaneously over a variety of scales, by creating a cluster tree. The tree is not a single set of clusters, but rather a multilevel hierarchy where clusters at one level are joined as clusters at the next higher level. This allows deciding what level or scale of clustering is the most appropriate so that the optimal data subset selection is scalable and flexible. It also gives the possibility to recursively combine with other cluster tree, so that large database can be split up to create several trees. The trees can then be combined to form the cluster tree with reasonable memory and computational requirements.

Given the defined generalized Levenshtein distance of any two entry pairs, the hierarchical clustering algorithm starts with all the entries in separate clusters and then repeatedly joins the two clusters that are most similar until there is only one cluster as shown in Figure 1.

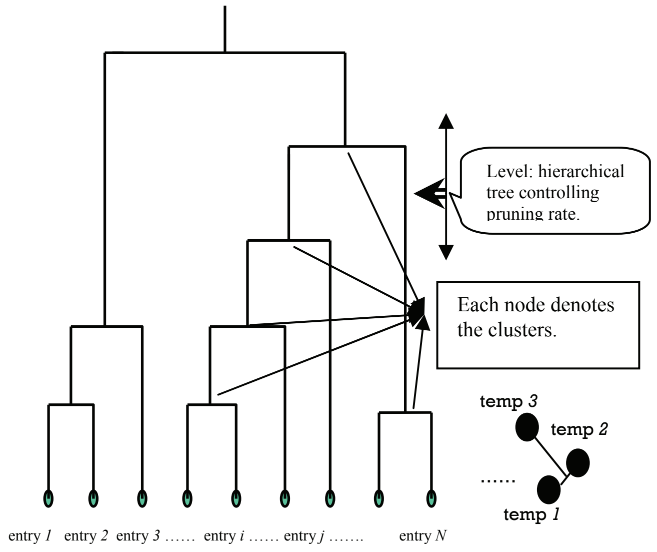

Figure 1. Hierarchical clustering tree

The history of merging forms a binary hierarchical clustering tree. The desired clustering of the entries can be obtained by cutting the hierarchical clustering tree at the desired level, and then each group of connected entries forms a cluster. Among the same-level clusters, it determines the two clusters that are the most similar based on the distance or similarity measure. The two determined clusters are replaced with a single merged cluster. The key step of generating a hierarchical clustering tree is repeatedly deciding which pair of clusters needs to be merged. As described above, the similarities between all the remaining clusters is first calculated. Then the two most similar clusters are grouped together to form a new cluster.

There are different types of approaches that can be used for measuring the similarity between clusters, for example single, complete and average group linkage. In this paper, the average linkage tends to join clusters with small and similar variances because it considers all the members in the cluster rather than just a single member as single and complete linkages. The distance between two clusters is the average distance between all the pairs as shown in Figure 2.

$$D_{IJ} = \frac{1}{N_I \cdot N_J} \sum_{i=1}^{I} \sum_{j=1}^{J} d(x_i, x_j) \qquad (5)$$

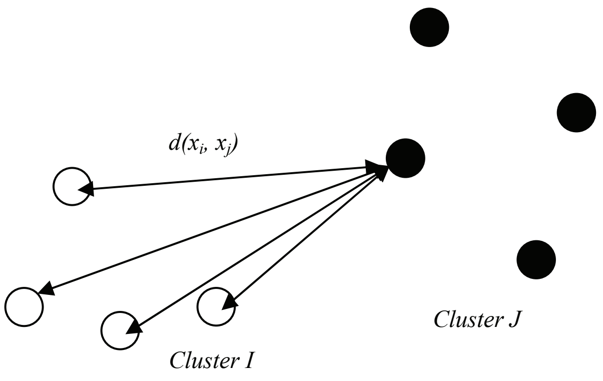

Figure 2. Average distance between two clusters in hierarchical clustering tree.

Finally, the cluster centroid that has the minimum total distance to all other entries within cluster is determined. The optimal subset is formed by all the cluster centroids.

3 DATA-DRIVEN SYLLABIFICATION

The development of speech synthesizers and speech recognizers often requires working with sub-word units such as syllables. We have earlier described a neural network based approach for the automatic assignment of syllable boundaries in [5]. In this paper, we revisit the topic and use this syllabification task for verifying the usefulness of the proposed text data selection approach.

Syllable is a basic unit of word studied on both the phonetic and phonological levels of analysis [2]. The syllable information can be described using grammars. The simplest grammar is the phoneme grammar, where a syllable is tagged with the corresponding phoneme sequence. The syllable structure grammar, on the other hand, divides a syllable into onset, nucleus and coda (ONC). The nucleus is an obligatory part that can be either a vowel or a diphthong. The onset is the first part of a syllable consisting of consonants and ending at the nucleus of the syllable. The part of a syllable that follows the nucleus forms the coda. The coda is constructed of consonants. The nucleus and the coda are combined to form the rhyme of a syllable. A syllable has a rhyme, even if it doesn't have a coda. In the syllable structure grammar, the consonants are assigned as onset or coda. The syllable structure grammar was used in this paper.

In the automatic syllabification task, the phoneme sequences are mapped into their ONC representations. The data-driven syllabification model is trained on the mapping information. In the decoding phase, given a phoneme sequence, the ONC sequence is first generated, and then the syllable boundaries are uniquely decided on the ONC sequence.

The basic neural network based ONC model presented in [5] is a standard multi-layer perceptron (MLP). The input phonemes are presented to the MLP network in a sequential manner. The network gives estimates of ONC posterior probabilities for each presented phoneme. In order to take the phoneme context into account, a number of phonemes on each side of the phoneme in question are also used as inputs to the network. Thus, a window of phonemes is presented to the neural network as input. The ONC neural network is a fully connected MLP, which uses a hyperbolic tangent sigmoid shaped function in the hidden layer and a softmax normalization function in the output layer. The softmax normalization ensures that the network outputs are in the range [0,1] and sum up to unity.

$$P_i = \frac{e^{y_i}}{\sum_{j=1}^{3} e^{y_j}}. \qquad (6)$$

In Equation (6), y_i and P_i denote the ith output value before and after softmax normalization. It has been shown in [1] that a neural network with softmax normalization will approximate class posterior probabilities when trained for one-out-of-N classification and when the network is sufficiently complex and trained to a global minimum. The ONC neural network is trained using the standard back-propagation (BP) algorithm augmented by a momentum term. Each phoneme with context and the corresponding ONC tag of the pronunciation make up one training example. Weights are updated in a stochastic on-line fashion.

The outputs of the ONC neural network approximate the ONC posterior probabilities corresponding to the centermost phoneme. The ONC sequence of a pronunciation is obtained by combining the network outputs for each individual phoneme in the pronunciation. Given a pronunciation with its phonemic representation, ONC tag of phoneme ph_i is given by

$$onc = \underset{onc_k}{\operatorname{argmax}} \{ P(onc_k \mid ph_{i-w}, ..., ph_{i+w}) \}, \qquad (7)$$

where $P(onc_k \mid ph_{i-w}, ..., ph_{i+w})$ is the network output corresponding to onc_k given the input phonemes $ph_{i-w}...ph_{i+w}$, and variable w denotes the phoneme window context size, respectively. The variable onc takes its values from the set [O N C].

4 EXPERIMENTS

A US English dictionary containing 5.4K entries is used for the experiments. Each entry of the dictionary includes a word, its pronunciation and the corresponding ONC label. The pronunciations and their ONC labels are extracted from the dictionary to form the training data. A reduced training data set is selected from the entire set by using the following methods:

- Decimation of the sorted dictionary (denoted as Decimation);
- Maximization of the objective function (denoted as Maximum Distance);
- Hierarchical cluster tree approach (denoted as Hierarchical);

The test set is constructed using the rest of data excluding the training set.

Figure 3 shows the string accuracy rate on test set vs. training data size, achieved using the different data optimization methods. The efficiency of the training data optimization approach can be studied by evaluating the generalization capability. The performance can be improved without increasing the size of the training set if the training data is well selected. The results clearly show that the proposed hierarchical and maximum distance techniques outperform the commonly used decimation method. The average improvement achieved using the proposed approach is 35.4%. The difference of performance between the hierarchical and the maximum distance schemes is marginal.

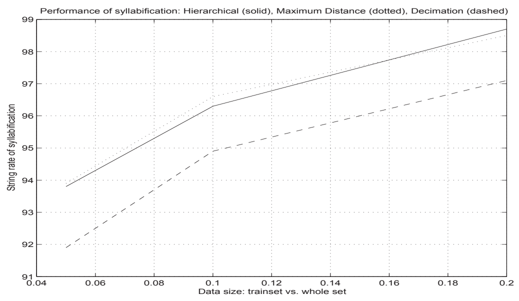

Figure 3. ONC accuracy on test set with different training set sizes among the three methods, with respect to the percentage of the selected subset size vs. whole data size.

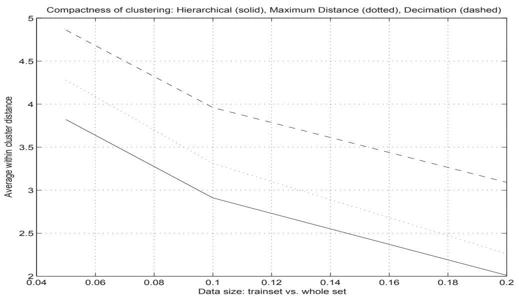

Figure 4. Compactness of text data clustering with different training set sizes among the three methods, with respect to the percentage of the selected subset size vs. whole data size.

The computational complexity of the maximum distance approach can be estimated as

$$C = \sum_{i=1}^{kN} (N-i) \cdot i = \frac{kN \cdot (kN+1) \cdot [(3-2k) \cdot N + 1]}{6} \propto O(N^3), \quad (8)$$

where N denotes the entire data size, and $M = kN$ stands for size of the selected data set. In comparison, the complexity for building a hierarchical tree is O(Nlog$N$). Clearly, the complexity of the maximum distance scheme is very high compared to the hierarchical scheme especially when optimal data size M is high.

As a second part of the experiments, the compactness of the clustered data was also studied. For the maximum distance and the decimation schemes, each entry in the selected set represents a cluster centroid. Suppose we have selected a data set SS={ss_1, ..., ss_M}, extracted from the data set S={s_1, ..., s_N}. For the ith entry in data set, its cluster is determined by searching the nearest centroid from the selected set.

$$c_i = \arg\min_{1 \le t \le M}(d(s_i, ss_t)) \quad (9)$$

Since all the data is clustered, the compactness can be calculated as the average intra-cluster distance across all the clusters. It represents the clustering/quantization error in such a manner that a low distance means a low clustering error.

Figure 4 illustrates the compactness with different training sizes, using the different data optimization methods. It is easy to see that the hierarchical scheme has more compact clusters than the maximum distance scheme. The decimation scheme has the largest clustering error. Thus, these results indicate that the proposed hierarchical method can cluster data with the highest accuracy, i.e. the selected data has better coverage and less error. Naturally, this explains the better performance achievable using data selection.

5 CONCLUSIONS

Text data selection for model training is a crucial, but often neglected, step in the development of speech and language processing systems. In this paper, we have proposed a new data selection approach based on hierarchical clustering. We have compared the proposed technique with the commonly used simple decimation and the previously developed maximum distance methods in terms of generalization capability, computational complexity and text data clustering error. Our experimental results obtained in the data-driven syllabification task show that the proposed approach is a very promising technique that makes it possible to carry out subset selection with good coverage, reduced complexity, low memory footprint, low clustering error and scalable using different levels of the hierarchical tree. The presented idea can be utilized in several applications that require training with text data, especially suitable for resources constraint embedded platforms, such as mobile devices.

6 ACKNOWLEDGEMENTS

This work was partly supported by the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement 213845 (EMIME project).

7 REFERENCES

[1] C. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, Oxford, UK, 1995.

[2] D. Kahn, *Syllable-Based Generalizations in English Phonology*, Doctoral Dissertation, Massachusetts Institute of Technology, USA, 1976.

[3] E. Ristad and P. Yianilos, "Learning String Edit Distance", *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol.20, pp.522-532, May, 1998.

[4] J. Suontausta, and J. Häkkinen, "Decision Tree Based Text-to-Phoneme Mapping for Speech Recognition," In *Proceedings of 6th ICSLP*, Beijing, China, 2000.

[5] J. Tian, "Data-Driven Approaches for Automatic Detection of Syllable Boundaries", in *Proceedings of 8th ICSLP*, Jeju Islands, Korea, 2004.

[6] J. Tian, J. Häkkinen, S. Riis, and K. Jensen, "On Text-Based Language Identification for Multilingual Speech Recognition Systems, In *Proceedings of 7th ICSLP*, Denver, USA, 2002.

[7] J. Tian, J. J. Nurminen, and I. Kiss, "Optimal Subset Selection From Test Databases", In *Proceedings of ICASSP*, Philadelphia, USA, 2005.