2015 IEEE 14th International Conference on Machine Learning and Applications

Model Shrinking for Embedded Keyword Spotting

Ming Sun
Amazon.com, Inc.
Cambridge, MA
Email: mingsun@amazon.com

Varun Nagaraja
Department of Computer Science
Univ. of Maryland, College Park, MD
Email: varun@umiacs.umd.edu

Björn Hoffmeister
Amazon.com, Inc.
Seattle, WA
Email: bjornh@a2z.com

Shiv Vitaladevuni
Amazon.com, Inc.
Cambridge, MA
Email: shivnaga@amazon.com

***Abstract*—In this paper we present two approaches to improve computational efficiency of a keyword spotting system running on a resource constrained device. This embedded keyword spotting system detects a pre-specified keyword in real time at low cost of CPU and memory. Our system is a two stage cascade. The first stage extracts keyword hypotheses from input audio streams. After the first stage is triggered, hand-crafted features are extracted from the keyword hypothesis and fed to a support vector machine (SVM) classifier on the second stage. This paper focuses on improving the computational efficiency of the second stage SVM classifier. More specifically, select a subset of feature dimensions and merge the SVM classifier to a smaller size, while maintaining the keyword spotting performance. Experimental results indicate that we can remove more than 36% of the non-discriminative SVM features, and reduce the number of support vectors by more than 60% without significant performance degradation. This results in more than 15% relative reduction in CPU utilization.**

Keywords*—*keyword spotting, feature selection, support vector merging

I. Introduction

Keyword spotting works to detect the words of interest in speech utterances, which has been an active research area in speech recognition for decades. One approach for keyword detection applies general large vocabulary continuous speech recognition (LVCSR) systems to decode the audio signal, with keyword searching conducted in the resulting lattices or confusion networks [1], [2], [3], [4]. These methods require relatively high computational resources for the LVCSR decoding, and are impractical for an embedded device.

Another widely used approach for keyword spotting builds hidden Markov models (HMM) for each keyword and non-keyword audio signal [5], [6], [7], [8]. The non-keyword audio signal includes other spoken words, background noise, etc. There can be one or more HMMs built to model the non-keyword audio characteristics, which are named filler models. Viterbi decoding is used to search the best path in the decoding graph, and the decoding output is further processed to make the decision on keyword presence. This approach can be extended to include discriminative information by incorporating a hybrid DNN-HMM decoding framework [9].

In recent years, with the burgeoning applications of deep learning techniques, some keyword spotting systems are built on DNN/RNN structures directly, without involving an HMM [10], [11], [12]. Those systems estimate the posteriors of keywords with context information, either by stacking frames within a context window for a DNN, or using an RNN. Then, posterior threshold tuning or smoothing is applied for decision making.

For our work, an embedded always-on keyword spotting system running on a device with real-time keyword detection, small memory footprint and low CPU utilization is of interest. We use a two-stage keyword spotting framework, with a keyword hypothesis extractor and an SVM classifier working together in a cascading manner. The first stage extractor runs on audio streams and extracts audio segments hypothesized to contain the pre-specified keyword. It is tuned for high recall. Given the hypothesized keyword audio segments from the first stage, hand-crafted features are computed. Those features are dumped to the second stage for decision-making. On the second stage an SVM classifier is applied on the features of the hypothesized keyword candidates, with the purpose of filtering out false accepts.

Due to limited computational resources on the device, it is important to improve the efficiency of our two-stage embedded keyword spotting system. This paper focuses on the optimization of the second stage SVM classifier. Two techniques are investigated for our experiments: feature selection and support vector merging. Feature selection refers to the process of selecting a subset of feature dimensions which contain most of the discriminative characteristics for the keyword, while support vector merging works to reduce the number of support vectors by merging similar ones for an existing SVM model.

The remaining part of this paper is organized as follows: Section II summarizes the two-stage keyword spotting system. Section III discusses the feature selection and support vector merging techniques used to optimize the second stage SVM classifier. Experimental results including keyword spotting performance and system CPU utilization with selected feature dimensions and merged SVM model are presented in Section IV. Section V is the conclusion.

II. System Overview

As shown in Figure 1, our embedded keyword spotting system is a cascade of two stages: an audio keyword hypothesis extractor as the first stage, and an SVM classifier as the second stage.

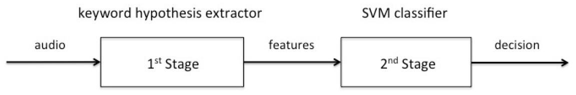


Fig. 1: Two-stage embedded keyword spotting system

978-1-5090-0287-0/15 $31.00 © 2015 IEEE
DOI 10.1109/ICMLA.2015.121

A. Stage One: Audio Keyword Hypothesis Extractor

For our keyword spotting system, the first stage consists of a DNN-HMM decoder. DNN acoustic models have shown superior performance over traditional Gaussian mixture model (GMM) models for speech recognition, and has become state-of-the-art in recent years [13]. For the DNN-HMM framework, HMM is used to model the sequential structure of speech, as in the GMM-HMM speech recognition system, while DNN predicts HMM states posterior information given input acoustic features. For our work, the layer-wise pre-training is applied to guarantee good convergence of the network.

The first stage extracts the audio segments which potentially contain the spoken keyword. The input acoustic feature for the first stage is computed as 20-dimensional log-filterbank energies (LFBE) for a $25ms$ audio window shifting at $10ms$. The input to the DNN consists of stacked LFBE features for a context window, with 20 frames before and 10 frames after. The DNN outputs posteriors for keyword phone states and background speech/non-speech states.

Let $f_{\mathcal{W}}$ denote the confidence score for audio segment $\mathcal{W}$ computed by the first stage keyword hypothesis extractor. A higher value of $f_{\mathcal{W}}$ indicates that it is more likely the keyword is spoken in $\mathcal{W}$. Note that the keyword spotting system is running on input audio streams, and the length of hypothesis segment $\mathcal{W}$ varies based on the the way the keyword is spoken. The first stage is triggered when $f_{\mathcal{W}} \geq f$, with f being a threshold. A D-dimensional feature vector $\mathbf{v}$ is extracted from $\mathcal{W}$ for the second stage classification. The first stage decision threshold f is tuned to achieve high recall, at a cost of relatively high false accepts, because any loss in recall on the first stage cannot be recovered from the second stage.

B. Stage Two: SVM Classifier

As mentioned in Section II-A, when the first stage is triggered, a fixed D-dimensional feature vector $\mathbf{v}$ is extracted from the hypothesized keyword audio segment $\mathcal{W}$, based on its acoustic features and the information extracted from the first stage. For our system the feature vector $\mathbf{v}$ is of dimension 71 ($D = 71$). The model shrinking techniques described in Section III are general approaches working for different types of input SVM features.

1) Features: The feature vector $\mathbf{v}$ for keyword hypothesized audio segment $\mathcal{W}$ includes the information from both segments and phones. Segment level features include the duration of segment, keyword likelihood score, normalized likelihood score and posterior for the keyword. Regarding the phone based features, we consider absolute and relative phone duration, phone log-likelihood, averaged phone/speech confidence scores, local context features such as left/right phone confidence, and entropy based features based on context phone log-likelihoods. There are also speech/non-speech scores computed for the context window before and after the keyword hypothesized audio segment $\mathcal{W}$.

2) SVM: The second stage runs an SVM classifier on $\mathbf{v}$ to make a keyword detection decision. SVM finds a hyperplane that separates data of different classes with the maximum margin [14], [15]. Kernels are used when the data are not linearly separable in the original feature space. Widely used kernels include linear kernel, polynomial kernel, RBF kernel etc. For our work, the RBF kernel shows better performance compared to other kernels. Thus, we use the RBF kernel.

III. Model Shrinking for SVM

Since embedded keyword spotting is limited by the computational resources on the device, it is important to develop the system to detect a keyword in real-time, at a low cost of CPU and memory. The computational complexity of the second stage SVM classifier is directly proportional to the number of features and the number of support vectors. Here we present two methods to optimize the second stage SVM classifier in our embedded keyword spotting system: the first technique is to select a subset of feature dimensions with most of the discriminative information, and the second technique is to merge close support vector pairs together. As a result, we get a reduced size SVM model which maintains the classification performance with a smaller number of feature dimensions and support vectors. This reduced size SVM also has lower complexity, which can help prevent overfitting.

A. SVM Feature Selection

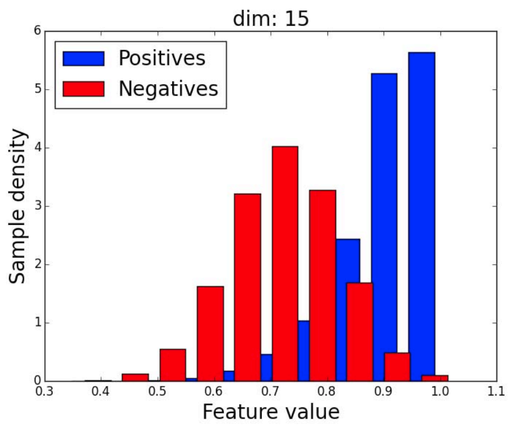
Feature selection selects a subset of feature dimensions, to maintain the majority of discriminative information, while removing redundancy and noise [16]. For our case, we select the SVM feature dimensions that include most of the discriminative information between the first stage extracted true positive and false positive keyword hypothesis. The discriminative information included in each feature dimension is measured based on Kullback-Leibler (KL) divergence [17]. This KL divergence based feature selection has been used for different applications [18], [19].

Let $\mathcal{C}$ denote a corpus with N hypothesized keyword audio segments extracted from the first stage. These N segments are labeled as $\mathcal{W}_1, \dots, \mathcal{W}_N$, which can be divided into two groups: the true positive segment group $\mathcal{C}_{tp}$ which consists of the segments with a keyword, and the false positive segment group $\mathcal{C}_{fp}$ which consists of segments with no keyword. For each segment $\mathcal{W}_n,\ n \in \{1, \dots, N\}$, a D-dimensional feature vector $\mathbf{v}_n$ is extracted. Feature selection selects a subset of feature dimensions maintaining most of the discriminative information for distinguishing $\mathcal{C}_{tp}$ and $\mathcal{C}_{fp}$.

For each dimension $d \in \{1, \dots, D\}$, let P_d and Q_d denote the distribution of feature values in $\mathcal{C}_{tp}$ and $\mathcal{C}_{fp}$, respectively. Examples of P_d and Q_d for a specific dimension d are shown on Figure 2, where the x-axis is the feature value and y-axis labels the sample density distribution. The red and blue colors represent true positive group $\mathcal{C}_{tp}$ and false positive group $\mathcal{C}_{fp}$. It can be observed that feature dimension 15 shown on Figure 2 includes more distinguish information to separate $\mathcal{C}_{tp}$ and $\mathcal{C}_{fp}$, compared to feature dimension 23.

We use KL divergence to measure the difference between $\mathcal{C}_{tp}$ and $\mathcal{C}_{fp}$ for each feature dimension. The KL divergence of Q_d from P_d is defined as

$$D_{KL}(P_d \parallel Q_d) = \int_{-\infty}^{+\infty} P_d(x) \ln \frac{P_d(x)}{Q_d(x)} dx. \tag{1}$$



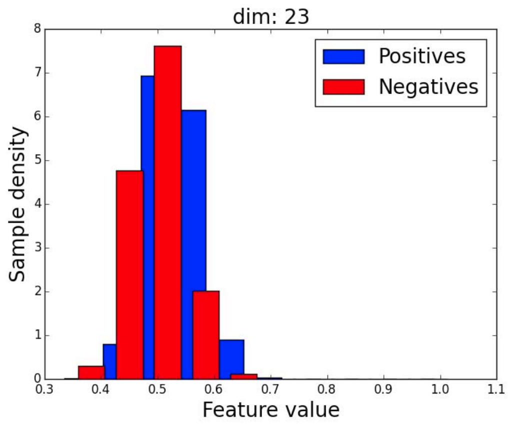


Fig. 2: Density examples for two feature dimensions (dim 15 and dim 23)

Since this is an asymmetric measure, we calculate both $D_{KL}(P_d \parallel Q_d)$ and $D_{KL}(Q_d \parallel P_d)$, and take the minimum of the two as a symmetric KL divergence measure. Only those dimensions with symmetric KL divergence above a threshold are retained. That is, given a threshold θ, we keep dimension d' when

$$\min(D_{KL}(P_{d'} \parallel Q_{d'}), D_{KL}(Q_{d'} \parallel P_{d'})) \geq \theta. \qquad (2)$$

The threshold θ can be tuned on a development set, which selects a value of θ to remove as many no-discriminative feature dimensions as possible, with no significant classification performance degradation. As a result, the new feature vector $\mathbf{v}'$ is of dimension D' $(D' \leq D)$ after feature selection. For our case, $D' = 45$, and examples of selected features include phone durations etc.

B. Support Vector Merging

We also merge support vector pairs which are close to each other, with the purpose of reducing the number of support vectors [20]. Alternatively, the support vector number can be reduced by changing slack variable penalty or kernel related parameters (e.g. γ for RBF kernel) for SVM training. However, our experiments with changing hyperparameters for SVM training show performance degradation in keyword spotting. Thus, for SVM training, we select the hyperparameters such as slack variable penalty and kernel parameters from a tuning step, choosing the values of these hyperparameters to optimize the classification performance on a development set. We fix these hyperparameters for SVM training. After that, we merge support vectors for the trained SVM to get a merged SVM model with fewer number of support vectors. As shown in Section IV, this reduces the SVM model size with no significant change in performance. The support vector merging algorithm used is based on [20]. Figure 3 shows the algorithm flowchart.

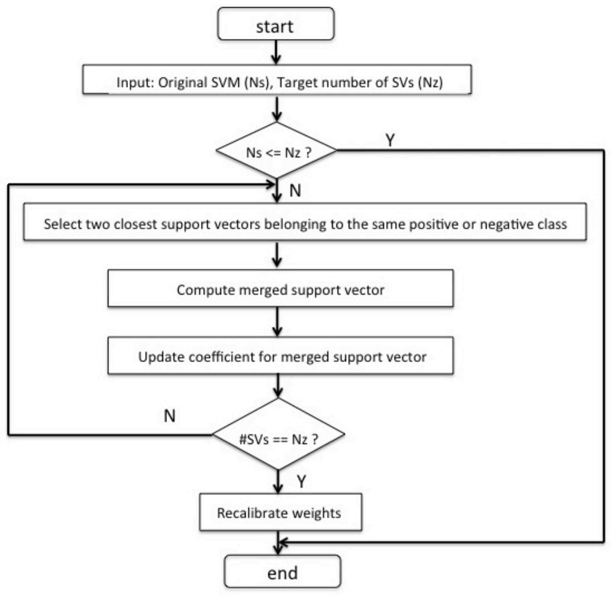

start
Input: Original SVM (Ns), Target number of SVs (Nz)
Ns <= Nz ?
Y
N
Select two closest support vectors belonging to the same positive or negative class
Compute merged support vector
Update coefficient for merged support vector
N
#SVs == Nz ?
Y
Recalibrate weights
end

Fig. 3: *Support vector merging algorithm*

Given the original trained SVM model with N_s support vectors and the target support vector number N_z for merging, the algorithm selects the closest support vector pair (SV_i, SV_j) belonging to the same positive or negative class based on Euclidean distance, and merges the pair to a single support vector SV' by interpolating SV_i, SV_j via an optimized weight. In more detail, let $\phi(SV_i), \phi(SV_j)$ represent the high-dimensional space mappings of support vectors SV_i, SV_j, and α_i, α_j be the corresponding weights for these two support vectors. The merging algorithm replaces $\phi(SV_i)$ and $\phi(SV_j)$ by

$$M = \frac{\alpha_i}{\alpha_i + \alpha_j}\phi(SV_i) + \frac{\alpha_j}{\alpha_i + \alpha_j}\phi(SV_j). \qquad (3)$$

As a result, the merged support vector SV' is computed by optimizing

$$\min_{SV'} ||M - \phi(SV')||. \qquad (4)$$

It can be shown that for the RBF kernel, the merged support vector is $SV' = kSV_i + (1-k)SV_j$, where the interpolation weight k is derived from optimizing an objective function. When the merged support vector SV' is computed, its weight α' for SV' also needs to be updated as

$$\alpha' = \frac{(\alpha_i + \alpha_j)M \cdot \phi(SV')}{||\phi(SV')||^2}. \qquad (5)$$

The merging process is run iteratively until the target number of support vectors is reached. As the last step, the weights for all remaining support vectors (either original support vectors or merged ones) are optimized globally. [21]

IV. Experimental Results

Our embedded keyword spotting system is run on a device. The device is activated when the keyword is spoken. To evaluate the keyword spotting performance, we use two metrics. The first one is the miss-rate which is defined as one minus recall. The second one is the false accept rate (FAR), which is the frequency of false positive detections.

A high-performance keyword spotting system targets a low miss-rate and low FAR. For our experiments, we train multiple SVM models with full feature dimensions and selected feature dimensions. The SVM model trained on selected feature dimensions is further shrunk via support vector merging. As a result, we have reduced size SVM models in selected feature subset, either with or without support vector merging. By comparing those reduced size SVM models with the baseline SVM trained on full feature dimensions with no support vector merging, we investigate the effects of feature selection and support vector merging on keyword detection accuracy and CPU utilization. Note that the detailed information about datasets, absolute evaluation numbers etc has been obscured in this paper due to confidentiality reasons. This paper compares systems with model shrinking techniques with baseline system in terms of relative difference in keyword spotting performance and CPU utilization.

A. Master Test Set

Our embedded keyword spotting system is evaluated on a master test set. The master test set consists of multiple test sets from distinct sources, e.g. some datasets include recordings of utterances from activated devices with embedded keyword spotting system, when these devices are exposed to Amazon employees who join an internal Beta group for device testing. Note that those utterances could either contain a valid keyword, or actually be false positives. We also record audio with/without keyword for evaluation purpose. Several background audio datasets are included in the master test set as well, mainly used to measure FAR in general background. For our experiments, we select a specific word as the keyword and build the keyword spotting system for it. The particular test set we have been using has tens of hours of audio in total with several thousands of instances of the selected keyword.

B. Accuracy Performance

For evaluating the accuracy of our keyword spotting system, we focus on a selected low miss-rate range we are interested in. At the same miss-rate within the range of interest, we compare the system with baseline SVM to systems with reduced size SVM, computing the relative difference in FAR for all the reduced size systems. We also select a system operating point (OP) with a specific miss-rate number within the range and show the relative difference in FAR for all the reduced size systems at the operating point.

The full SVM feature set has 71 dimensions ($D = 71$). We use the keyword spotting system with SVM trained on the full feature set with no support vector merging as the baseline. The baseline SVM model has approximately $15.9k$ support vectors. With the application of the KL divergence based feature selection procedure described in Section III-A, only 45 dimensions are kept ($D' = 45$). This counts for more than 36% reduction in feature dimensions. The SVM trained on selected feature dimensions has approximately $15.7k$ support vectors. We apply support vector merging technique described in Section III-B on the $15.7k$ support vector SVM trained on selected feature dimensions, reducing the support vector to $10k$ and $6k$, respectively. For the keyword spotting systems with these three reduced size SVMs, Table I shows the relative difference in FAR given the same miss-rate.

TABLE I: *Within the selected miss-rate range of interest, the maximum, minimum, and operating point associated FAR relative difference (given the same miss-rate) for the keyword spotting systems with reduced size SVM compared to the keyword spotting system with baseline SVM.*

Shrinked models (select dim)	max	min	OP
original $15.7k$ SVM	+1.9%	-2.5%	+0.1%
merged $10k$ SVM	+1.4%	-2.1%	+0.5%
merged $6k$ SVM	+3.4%	-1.7%	+0.2%

The three rows of Table I show the original size SVM (approximately $15.7k$ support vectors), merged $10k$ support vector SVM and merged $6k$ support vector SVM, respectively. All three SVMs are trained on selected feature dimensions. We compute the relative difference in FAR for these three keyword spotting systems with reduced size SVM models compared to the baseline system with SVM trained on full feature set and with no support vector merging. For the selected miss-rate range of interest, the first two columns of Table I show the maximum and minimum relative difference in FAR with the same miss-rate within the selected range, respectively. The last column shows the FAR relative difference for the selected operating point.

It can be observed that when the SVM is trained on selected 45 feature dimensions and the support vectors are merged to $6k$, the keyword spotting system still shows similar performance compared to the baseline system with original size SVM trained on full feature dimension with no support vector merging, in the selected miss-rate range of interest. The FAR difference for the selected miss-rate range is within our tolerance region. At the selected operating point, all the four systems with different feature dimensions and support vector numbers perform with no significant difference.

C. CPU Utilization

To measure the CPU reduction by taking advantage of second stage SVM model shrinking techniques, we run our embedded keyword spotting system on the device on a selected set of audio files, and measure the 1-second window average CPU usage. The dataset used contains several hours of audio for intensive CPU profiling purpose.

Figure 4 shows the relative CPU percentage for all the four systems described in Section IV-B, where the x-axis represents CPU profiling audio percentile and the y-axis represents the

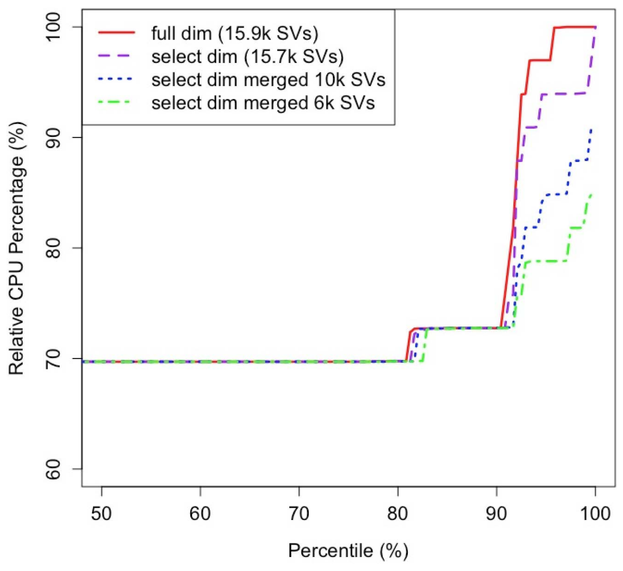


Fig. 4: *CPU utilization for the baseline keyword spotting system and three systems with reduced size SVM. The keyword spotting system CPU utilization is shown as relative CPU percentage. It is plotted based on the percentile of the CPU profiling audio files.*

relative on-device CPU percentage for the embedded keyword spotting system. The maximum CPU usage (P100 number for the baseline system) is scaled to 100%. The CPU usage for all the four systems are scaled in the same way to show relative CPU percentage. It can be observed that all four systems show similar CPU usage up to the 90% percentile. The feature selection and support vector merging show significant CPU reduction in the high percentile region. As expected, the baseline full feature dimension SVM with no support vector merging (red solid curve) has the largest CPU usage among all four systems. The CPU utilization is reduced by training the SVM on selected feature dimensions (purple dashed curve), and further reduced by merging the selected feature dimension SVM to $10k$ (blue dotted curve) and $6k$ (green dotdash curve) support vectors.

In our CPU utilization measurement, there is a constant CPU consumption from the first stage keyword hypothesis extractor. The SVM classification on second stage runs only when the first stage is triggered by accepting the keyword hypothesis segment. This means the SVM stage directly contributes to the CPU spikes and system latency. As a result, shrinking the second stage SVM model helps bring down CPU spikes and reduce system latency. Applying a reduced size SVM model also reduces memory usage for the keyword spotting system running, though for our case memory is not a major concern.

The detailed CPU utilization relative reduction information for our keyword spotting systems with reduced size SVM is shown in Table II. As described in Section IV-B, the baseline keyword spotting system uses the original size SVM in full feature dimension. The three rows of Table II include the CPU relative reduction information for the systems with original size SVM, merged $10k$ support vector SVM, and merged $6k$ support vector SVM, respectively. All three SVMs in Table II are trained on selected feature dimensions. These three reduced size systems are compared to the baseline SVM system. The four columns are for the 50, 90, 99 and 100 percentile. It can be observed that when we select feature dimensions including most of the discriminative information, and merge the SVM to $6k$ support vectors, the CPU utilization is significantly reduced in the high percentile region, with 18.2% and 15.1% relative reduction for P99 and P100, respectively.

TABLE II: *CPU utilization relative reduction for our keyword spotting system with reduced size SVM. Table I shows that at the selected operating point, all systems perform with no significant difference for keyword spotting.*

Shrinked models (select dim)	P50	P90	P99	P100
original $15.7k$ SVM	0%	0%	-6.0%	0%
merged $10k$ SVM	0%	0%	-12.0%	-9.1%
merged $6k$ SVM	0%	0%	-18.2%	-15.1%

V. Conclusions

In this paper we have presented our on-device keyword spotting system. It is a two-stage embedded system, with a cascading structure of an audio keyword hypothesis extractor and an SVM classifier. This paper focuses on how to optimize the second stage SVM classifier, i.e., how to shrink the model to reduce computational cost, with no significant performance degradation. Two techniques are discussed: feature selection and support vector merging. Experimental results show that our keyword spotting system built for a selected keyword detection maintains the performance when more than 36% of non-discriminative SVM feature dimensions are removed, plus more than 60% reduction in support vector number. As a result, more than 15% of relative CPU reduction can be achieved in the high percentile region.

VI. Acknowledgements

We gratefully thank Spyros Matsoukas for suggestions on this work, and Aparna Khare, Baiyang Liu, George Tucker, Gengshen Fu and Sankaran Panchapagesan for contributing to building the keyword spotting system.

References

[1] Miller, D.R., Kleber, M., Kao, C.L., Kimball, O., Colthurst T., Lowe, S.A., Schwartz, R.M., and Gish, H., "Rapid and accurate spoken term detection", in *8th Annual Conference of the International Speech Communication Association*, 2007.

[2] Parlak, S. and Saraclar, M., "Spoken term detection for Turkish broadcast news", in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5244-5247, 2008.

[3] Chen, G., Yilmaz, O., Trmal, J., Povey, D. and Khudanpur, S., "Using proxies for OOV keywords in the keyword search task", in *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pp. 416-421, 2013.

[4] Tsakalidis, S., Hsiao, R., Karakos, D., Ng, T., Ranjan, S., Saikumar, G., Zhang, L., Nguyen, L., Schwartz, R. and Makhoul, J., "The 2013 BBN vietnamese telephone speech keyword spotting system", in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 7829-7833, 2014.

[5] Rose, R.C. and Paul, D.B., "A hidden Markov model based keyword recognition system", in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 129-132, 1990.

[6] Wilpon, J.G., Rabiner, L., Lee, C.H. and Goldman, E.R., "Automatic recognition of keywords in unconstrained speech using hidden Markov models", *IEEE Transactions on Acoustics, Speech and Signal Processing*, 38(11):1870-1878, 1990.

[7] Wilpon, J.G., Miller, L.G. and Modi, P., "Improvements and applications for key word recognition using hidden Markov modeling techniques", in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 309-312, 1991.

[8] Benayed, Y., Fohr, D., Haton, J.P. and Chollet, G., "Confidence measures for keyword spotting using support vector machines", in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 588-591, 2004.

[9] Ketabdar, H., Vepa, J., Bengio, S. and Bourlard, H., "Posterior based keyword spotting with a priori thresholds", in *Interspeech*, 2006.

[10] Fernndez, S., Graves, A. and Schmidhuber, J., "An application of recurrent neural networks to discriminative keyword spotting", in *Artificial Neural Networks-ICANN*, pp. 220-229, 2007.

[11] Chen, G., Parada, C. and Heigold, G., "Small-footprint keyword spotting using deep neural networks", in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4087-4091, 2014.

[12] Baljekar, P., Lehman, J.F., and Singh, R., "Online word-spotting in continuous speech with recurrent neural networks", in *IEEE Spoken Language Technology Workshop (SLT)*, 2014.

[13] Hinton, G., Li, D., Yu, D., Dahl, G.E., Mohamed, A.R., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T.N. and Kingsbury, B., "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups", *IEEE Signal Processing Magazine*, 29(6): 82-97, 2012.

[14] Cortes, C. and Vapnik, V., "Support-vector networks", *Machine Learning*, 20(3): 273-297, 1995.

[15] Chang, C.C. and Lin, C. J., "LIBSVM: a library for support vector machines", *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.

[16] Guyon, I. and Elisseeff, A., "An introduction to variable and feature selection", *The Journal of Machine Learning Research*, 3: 1157-1182, 2003.

[17] Kullback, S. and Leibler, R. A., "On information and sufficiency", *The Annals of Mathematical Statistics*, 22: 79-86, 1951.

[18] Schneider, K. M., "A new feature selection score for multinomial naive Bayes text classification based on KL-divergence", in *42nd Annual Meeting of the Association for Computational Linguistics*, pp. 186-189, 2004.

[19] Noda, T., Yano, Y., Doki, S. and Okuma, S., "Adaptive emotion recognition in speech by feature selection based on KL-divergence", in *IEEE International Conference on Systems, Man and Cybernetics*, vol. 3, pp. 1921-1926, 2006.

[20] Nguyen, D. and Ho, T., "An efficient method for simplifying support vector machines", in *22nd International Conference on Machine learning*, pp. 617-624, 2005.

[21] Scholkopf, B., Mika, S., Burges, C.J., Knirsch, P., Muller, K., Ratsch, G. and Smola, A. J., "Input space versus feature space in kernel-based methods", *IEEE Transactions on Neural Networks*, 10(5): 1000-1017, 1999.