Distributed analytics for audio sensing applications

David Wood[a], Shiqiang Wang[a], Theodoros Salonidis[a], Dave Conway-Jones[b], Bongjun Ko[a], and Graham White[b]

[a]IBM T.J. Watson Research Center, 1101 Kitchawan Rd, Yorktown Heights, NY, USA
[b]Emerging Technology, IBM UK, MP137, IBM Hursley Park, Winchester, SO21 2JN, UK

ABSTRACT

A wide array of military and commercial applications rely on the collection and processing of audio data. One approach to perform analytics and machine learning on such data is to upload and process them at a central server (e.g., cloud) which offers abundant processing resources and the ability to run sophisticated machine learning models and analytics on the audio data. This approach can be inefficient due to the low bandwidth and energy limitations of mobile devices as well as intermittent connectivity to a central collection point such as the cloud. It is also problematic as audio data are often highly sensitive and subject to privacy constraints. An alternative approach is to perform audio analytics at edge of the network where data is generated. The challenge in this approach is the requirement to perform analytics subject to resource constraints which limit performance and accuracy of predictive analytics. In this paper, we present a system for performing predictive analytics on audio data, where the training is executed on the cloud and the classification can be executed at the edge. We present the design principles and architecture of the system, and quantify the performance tradeoff of executing analytics at contemporary edge devices versus the cloud.

Keywords: Audio analytics, machine learning, edge computing

1. INTRODUCTION

Audio sensing applications have become increasingly popular in the recent years, with applications in both commercial and tactical scenarios.[1–8] However, the large amount of data generated from continuous audio monitoring causes a significant amount of communication overhead. In order to minimize bandwidth usage, systems should ideally only report relevant and significant events. Furthermore, what is classified as relevant may vary depending on the application scenario, and the significance threshold can also evolve. Such systems thus need to be able to perform local classification of sounds, but also be adaptable in order to maintain relevance and not cause information overload.

With the increasing availability of sophisticated low power processing capability it is now possible to envisage adding these capabilities to man-portable equipment, such as a smart-phone or military equivalent platform. This would enable the continuous classification of background audio signals, potentially providing the user with alerts about vehicle movements, incoming fire, and so on. This information can then also be shared with other personnel as needed.

In this paper, we present an audio sensing system that has both cloud and edge components. In the system, the machine learning models for acoustic analytics are trained in the cloud. The classification of real time audio input can be performed either in the cloud or on the edge, where the edge can be a mobile phone or an embedded system such as Raspberry Pi. We present the architecture and the design of the system, followed by experimental evaluation of the model accuracy and the training and classification times.

2. SYSTEM DESIGN

2.1 Requirements

The architecture for our end-to-end system is guided by the following requirements.

- *Adaptive modeling:* We would like to incorporate ensemble modeling techniques that can adapt themselves during training to optimize for a particular domain. For example, some acoustic environments might be better modeled using a simple fast Fourier transform (FFT) feature extraction over large sub-windows, and others might do better with mel-frequency cepstral coefficients[9] (MFCC) coupled with velocity and acceleration processing on smaller sub-windows. In addition, different machine learning algorithms may also be better suited to different domains.
- *Edge- and cloud-based operations:* Our training and classification services must be available as a service over the network, initially to support edge-based operations, but also as general services that can be used from any client. The ability to run on the edge requires our models to be of limited size regardless of the amount of training data.
- *Incremental learning:* We realize that our initial models will not be as accurate as we would like and so we need to be able to capture additional data that can be labeled and trained back into our operational systems.
- *Data scalability:* The training data can consist of gigabytes, terabytes and even petabytes of audio samples. For example, 10 hours of PCM audio data, a relatively small dataset, can range from 1 to over 6 GB depending on the audio quality. We need to be able to train our models without having to have all the data in memory at the same time.
- *High performance:* The support for ensemble modeling techniques requires us to be efficient and design novel approaches to avoid computational redundancy. In addition, we want to be able train models "on the fly," which generally means a training time that is a small percentage (1%–5%) of the training data length for our shallow models. In addition, we need to be able take advantage of different computing environments (e.g., Spark, single VM, etc.) and architectural capabilities (e.g., multiple cores, GPUs, etc.).

2.2 Architecture

The complete system architecture to meet these requirements is depicted in Figure 1.

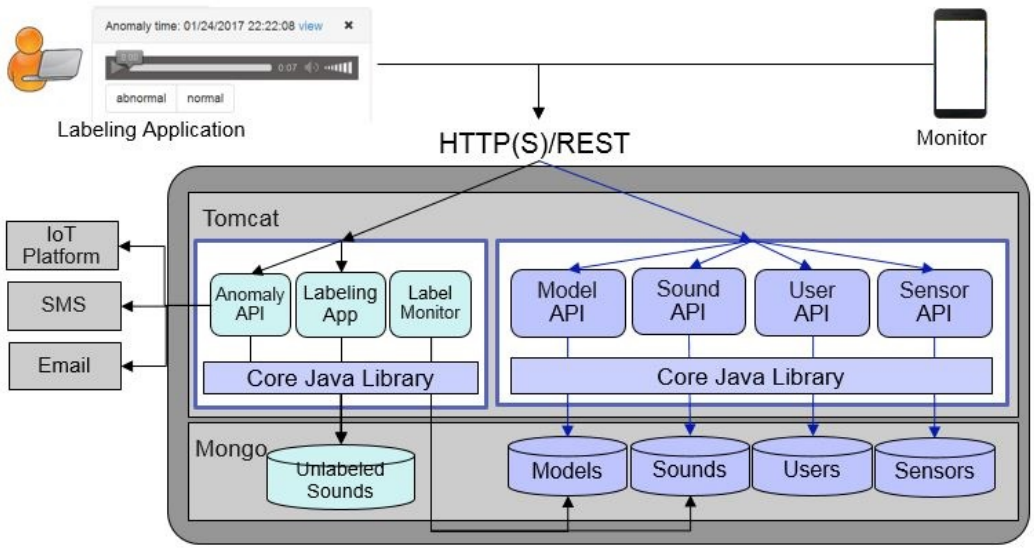

Figure 1. End-to-end system architecture.

On the right side of Figure 1 is a core set of application-independent services that include the following functionalities:

- Capture and curating training data;

- Train, storing and exporting models;
- Classifying unlabeled acoustic data using stored models;
- Manage user profiles and their devices, including notification preferences.

These sets of services are core to any machine learning platform and serve as the core services for both data curation and classifier training and use. Models are exported to a Java-based edge client using Java serialization of the trained model. This together with the limited size of our models enables us to run classifiers on edge devices such as Android and Raspberry Pi.

On the left side of Figure 1 are application-specific services for:

- Capturing unrecognized/anomalous sounds;
- Labeling the unlabeled sounds;
- Monitoring sound label updates for migration to training storage and model retraining;
- Notifying users and IBM's MQTT-based IoT platform.

These services altogether enable the incremental learning we seek to provide. By capturing outliers (i.e., unrecognized sounds or sounds with low classification confidence score), allowing the user to confirm or relabel the sounds, migrating the newly labeled sounds to training storage and retraining models, we are able to provide a system that grows its understanding of the acoustic environment over time.

2.3 Interfaces

In order to support efficient implementation of adaptive/ensemble modeling, we define a feature extraction pipeline for training and classification as shown in Figure 2.

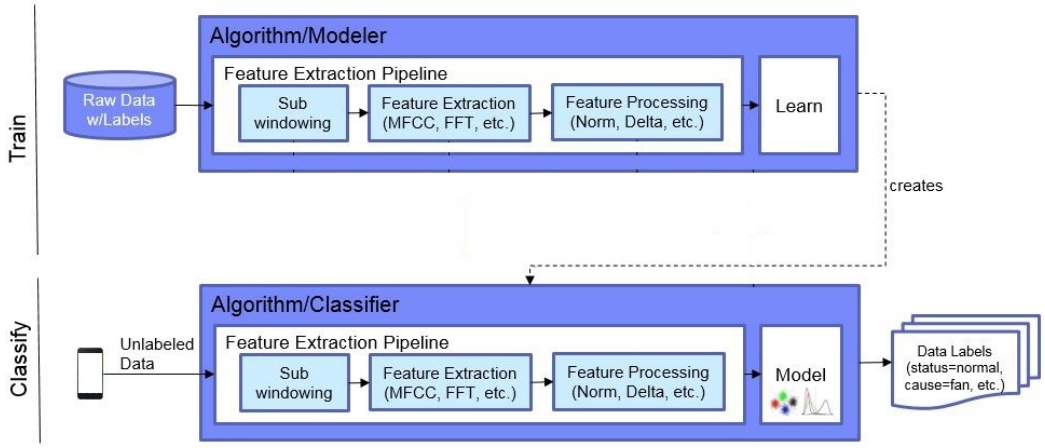

Figure 2. Feature extraction pipeline for training and classification.

At the core of the server architecture is our Java library which provides support for data scalability, training and classification performance, edge classification and adaptive modeling. The library is based around key classes/interfaces related to the feature extraction pipeline shown in Figure 2 as follows:

- *IDataWindow* holds the raw data used for both training and classification. This is a generic class enabling any type and shape of data, although we have focused on arrays of scalar data (e.g., double[]).
- *ILabeledDataWindow* pairs an IDataWindow with a set of one or more labels (as Java Properties).

- *IClassifier* represents a machine learning algorithm that can be trained on a stream of ILabeledDataWindows and provide classifications of IDataWindows after training.

Data scalability is primarily enabled by defining the training method as train(Iterable<ILabeledDataWindow> data) thereby requiring all algorithms to train on streaming data. Most algorithms require the extraction of features from the raw data and can benefit from extracting such features on sub-windows of the raw data. We further define the following in support of feature extraction:

- *IFeature* is a simple extension of IDataWindow to hold the feature data extracted from an IDataWindow.
- *ILabeledFeature* pairs an IFeature with a set of labels taken from the ILabeledDataWindow used to generate the IFeature.
- *IFeatureExtractor* provides a stateless operation to extract feature data from an IDataWindow. Examples of this include FFT and MFCC.
- *IFeatureProcessor* provides a stateless operation to transform a set of IFeature. Examples of this include normalization and velocity and acceleration computations across the spectrogram of features.

2.4 Machine Learning Algorithms

The machine learning algorithms used in the system include shallow models, such as nearest neighbor and Gaussian mixture model[10] (GMM), and deep models, such as convolutional neural networks.[11] The system also includes an ensemble algorithm[12] that evaluates multiple combinations of machine learning models, sub-window sizes, feature extractors and feature processors, and selects one or multiple of such combinations for further training. The purpose of the ensemble is to allow the system to adapt to different sound domains, where the best combination of algorithms may be different for different sound domains. All the models are trained on the cloud. The shallow models (and ensembles thereof) can run at the edge to perform classification of locally captured acoustic data samples.

3. EVALUATION

We evaluated the training and classification time and accuracy of models supported by our architecture using the Urban8K dataset defined by New York University.[13] The Urban8K data set consists of 8.7 hours of audio taken from multiple sources and containing eight different classes. The models we tested using this data set are defined in the table shown in Figure 3. The ensemble model contains different variants of GMM and nearest neighbor models, that are paired with different configurations of sub-windowing, feature extraction and processing. There are 7 models (combination of machine learning algorithm, sub-windowing, feature extractor and feature processor) configured in the training phase of the ensemble. The ensemble picks the top-3 models that give the best accuracy in K-fold evaluation on the training data set. It then trains these three models on

Algorithm	Sub-windowing	Feature Extractor	Feature Processor	Simple Name (used in charts)
Nearest Neighbor (Lp distance with p = 0.5)	100 ms, rolling	MFCC (20)	None	LpNN
GMM, 10 Gaussians	100 ms, rolling	MFCC (20)	None	GMM
GMM, 8 Gaussians	40 ms, 20 ms sliding	MFCC (40)	Velocity, acceleration	GMM/Delta
Ensemble 4 GMMs, 3 Nearest Neighbors	Various sliding	MFCC (20), MFCC (40)	5 None, 2 Velocity + acceleration	Ensemble

Figure 3. Models tested for performance and accuracy.

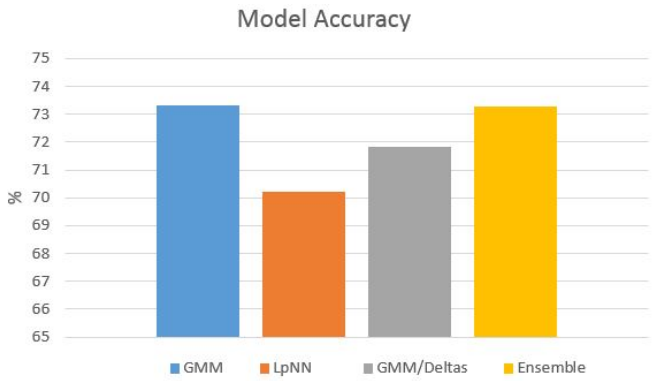

Figure 4. Models tested for performance and accuracy.

Platform	Model	CPU	Memory
Cloud	IBM x3650 M3	Intel Xeon 5650, 2.67 GHz, 12 cores	32 GB
Edge	Raspberry Pi 3 Model B Rev 1.2	ARM v7, 1.2 GHz, 4 cores	1 GB

Figure 5. Hardware platforms used for evaluating performance.

the entire training data set, and uses these trained models during classification both in the cloud and at the edge. The final classification result is obtained using a weighted-majority voting of the predictions from the three classifiers.

We first consider the accuracy of these different models. To evaluate the accuracy we defined 10 disjoint partitions by randomly placing each sound into one of the 10 partitions. For each set of 9 partitions, we trained the models and then tested their accuracy on the held out partition. We then averaged the 10 accuracy values for each model to get the results shown in Figure 4. All get over 70% accuracy and we can see that the ensemble did find and use the model with best accuracy.

We then consider training and classifications times for the various algorithms. The training times are evaluated on a cloud-equivalent machine and the classifications times are evaluated on both a cloud-equivalent and edge-class machine, as shown in the table in Figure 5. Both platforms used an Oracle 1.8 Java runtime.

First, we consider the training times of the various algorithms on the cloud platform. When evaluating the training time, we scale our times to the amount of training data so that our evaluations present the training time as a percentage of the overall training data length. These training time percentages are shown in Figure 6 and show that we can train all of the models in a fairly small percentage of the training data length, ranging from 0.25% to 2.75%. Feature extraction and processing consume most of the time in training and this is exhibited by the change in training time between the GMM and the GMM/Delta models. The latter uses twice the number of windows due to its use of overlapping windows and feature processing to compute the feature deltas. These deltas triple the length of each feature vector. Between the two we get a 6× change in complexity which roughly matches the change in performance.

Next, we evaluated the classification times of the various models on both the cloud and the edge platform. The results are shown in Figure 7. The first thing to notice is that the pattern of model classification performance across the models is similar on both platforms. We do see a roughly 16× degradation in performance between the cloud and edge platforms. The difference in CPU speed and the number of cores explains a 6.7× performance difference. The remaining 2.4× difference is likely due to bus speeds and the 64-bit vs 32-bit architecture of the two platforms.

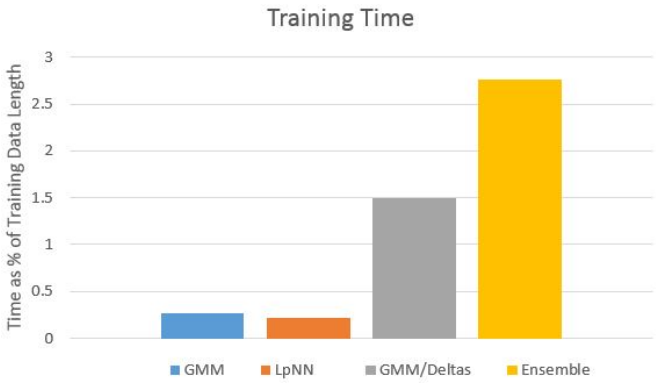

Figure 6. Cloud-based training time for different models as a percentage of training data length.

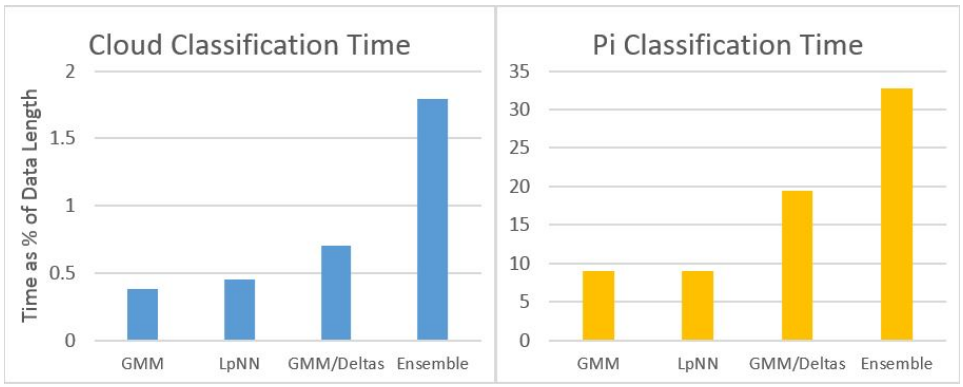

Figure 7. Cloud-based and Raspberry Pi-based classification time for different models as a percentage of training data length.

4. CONCLUSION

In this paper, we have presented a system for audio sensing. The system is distributed across the cloud and the edge. It allows the training of sound classification models in the cloud. The models can then be downloaded to the edge for the classification of new sounds. Experimentation results have shown that the edge classification time using various models of the system is less than 35% of the time length of the sound data, indicating that real-time classification at the edge is feasible.

ACKNOWLEDGMENTS

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

REFERENCES

[1] Verma, D., Ko, B. J., Wang, S., Wang, X., and Bent, G., "Audio analysis as a control knob for social sensing," in [*Proceedings of the 2nd International Workshop on Social Sensing*], 75–80, ACM (2017).

[2] Ko, B. J., Ortiz, J., Salonidis, T., Touma, M., Verma, D., Wang, S., Wang, X., and Wood, D., "Acoustic signal processing for anomaly detection in machine room environments: Demo abstract," in [*Proceedings of the 3rd ACM International Conference on Systems for Energy-Efficient Built Environments*], 213–214, ACM (2016).

[3] Lane, N. D., Georgiev, P., and Qendro, L., "Deepear: Robust smartphone audio sensing in unconstrained acoustic environments using deep learning," in [*Proc. of ACM UbiComp '15*], 283–294, ACM, New York, NY, USA (2015).

[4] Larson, E. C., Lee, T. J., Liu, S., Rosenfeld, M., and Patel, S. N., "Accurate and privacy preserving cough sensing using a low-cost microphone," in [*Proc. of ACM UbiComp '11*], 375–384, ACM, New York, NY, USA (2011).

[5] Lu, H., Pan, W., Lane, N. D., Choudhury, T., and Campbell, A. T., "Soundsense: scalable sound sensing for people-centric applications on mobile phones," in [*Proceedings of the 7th international conference on Mobile systems, applications, and services*], 165–178, ACM (2009).

[6] Murty, R. N., Mainland, G., Rose, I., Chowdhury, A. R., Gosain, A., Bers, J., and Welsh, M., "Citysense: An urban-scale wireless sensor network and testbed," in [*2008 IEEE Conference on Technologies for Homeland Security*], 583–588, IEEE (2008).

[7] Solomon, L. and Tran-Luu, D., "Tracking targets of interest via acoustics," tech. rep., ARMY RESEARCH LAB ADELPHI MD (2007).

[8] Tenney, S., Mays, B., Hillis, D., Tran-Luu, D., Houser, J., and Reiff, C., "Acoustic mortar localization system-results from oif," tech. rep., ARMY RESEARCH LAB ADELPHI MD (2004).

[9] Han, W., Chan, C.-F., Choy, C.-S., and Pun, K.-P., "An efficient MFCC extraction method in speech recognition," in [*Proc. of IEEE International Symposium on Circuits and Systems, 2006*], IEEE (2006).

[10] Shalev-Shwartz, S. and Ben-David, S., [*Understanding machine learning: From theory to algorithms*], Cambridge university press (2014).

[11] Goodfellow, I., Bengio, Y., and Courville, A., [*Deep Learning*], MIT Press (2016). http://www.deeplearningbook.org.

[12] Polikar, R., "Ensemble based systems in decision making," *IEEE Circuits and systems magazine* **6**(3), 21–45 (2006).

[13] Salamon, J., Jacoby, C., and Bello, J., "A dataset and taxonomy for urban sound research," in [*2014 ACM International Conference on Multimedia*], 1041–1044, ACM (2014).