SEVENTH FRAMEWORK PROGRAMME

Networked Media

Specific Targeted Research Project

SMART

(FP7-287583)

Search engine for MultimediA environment generated contenT

D3.3.1 Audio Signal Processing Prototypes

Due date of deliverable: 01-04-2013

Actual submission date: 11-06-2013

Start date of project: 01-11-2011 Duration: 36 months

Summary of the document

Code:	**D3.3.1 Audio Signal Processing Prototypes**
Last modification: **State:**	2013-06-11 Final

Participant Partner(s):
Author(s):
Fragment:
	IBM
	Zvi Kons (IBM), Orith Toledo-Ronen, Shay Ben-David, Ron Hoory
	NO

Audience:	☒ public ☐ restricted ☐ internal
Abstract:	*Report on the first release of the audio prototypes related to tasks T3.5 and T3.6.*
Keywords:	• *Audio classification* • Speech transcription
References:	DoW; DPA

Table of Contents

1 Executive Summary

1.1 Scope

The SMART system is designed to provide access to information that is currently not accessible to the ordinary search engines. In this report we deal with information in multimedia form. Although our world and the internet is filled with multimedia, it is currently mostly hidden from the search engine unless it was manually annotated (if the form of tags, titles and textual descriptions).

This deliverable deals with two types of multimedia information: the first one is in the form of audio events extracted from audio samples. The second one is of social data in the form of user voice messages. The two prototypes presented in this report are designed to deal with this type of information.

The first part of this report describes the audio classifier we've developed for this project. This audio classifier can analyze audio samples and provide information about the audio events inside them. Our main advance in this area is the successful usage for audio classification of the deep neural network (DNN) classifier, which has recently proven to be very useful for speech and image classification [8][9][13].

In the second part of the report we discuss our second prototype for speech processing. This prototype allows users to update the system with information in spoken form. Unlike social media such as Twitter and Facebook, which is limited to textual input, this will provide crowdsourcing functionality with voice interface.

1.2 Audience

The information presented in this report would be mostly of interest for the SMART project members. From this report they can learn the following:

- What kind of information is generated by the audio & speech components
- How those components integrate with the rest of the system.
- How the information is extracted from the audio and the speech.

Some of the information presented here is also relevant to the SMART open source community. However, since this report is published with restricted access, the relevant information is also published on the SMART open-source website.

1.3 Summary

This report presents the release of the first version for two prototypes:

Audio event classifier: we present our work to create an audio event classifier which is based on Deep Neural Network (DNN). We discuss our advances in this area and compare our classifier to other similar classifiers known in the literature. We show that our classifier outperform the others.

Speech processing: we describe the first version of the speech processing component which includes a speech transcription module.

1.4 Structure

The audio classifier is described in section 2. Section 3 presents the speech processing component. The conclusions are discussed in section 4.

2 Audio classification

2.1 Introduction

Extracting various types of information from speech is a topic for considerable amount of research in the worldwide research community. On the other hand, there is much less work on audio events analysis. In the SMART project we work on a framework for searching and analyzing of new types of information from multimedia data of the physical world. As part of this project, we would like to know what we could learn about the world around us using audio data.

The scope of this work is to extract some basic information from the audio sources, such as an indication of the presence of some audio events. This could be for example the information that an audio segment contains traffic or crowd noises. This audio event detection information can later be fused with additional data such as input from other audio and video sensors to produce higher reasoning.

There are several different approaches for audio event classification. The basic flow in most of them is to extract spectral based features and then apply some classification tools to distinguish between the classes. Simple approaches follow speech processing techniques with Mel-Frequency Cepstral Coefficients (MFCC) features and GMM based classification [1]. Other use features such as spectral features, Linear Predictive Coding (LPC), perceptual features and mixtures of features. Different classifiers are also used where the common ones are: nearest neighbours, Support Vector Machine (SVM), Radial Basis Function Neural Networks (RBFNN) and random forest [2],[3],[4],[5]. More complex techniques use audio pattern matching as in [6],[7].

Since each article tests its work with different audio data sets, different audio classes and different protocols, it is very hard to compare the different techniques and to determine which one is the best. Publicly available audio databases which can be used to evaluate various algorithms are common for the speech processing world but none are available for this type of audio event classification task. Because of the delays in the data collection tasks (D3.4) we were not able to acquire the data as originally planned. Our first task was therefore to acquire data from other sources.

Recently a significant improvement in speech recognition was achieved by using Deep Neural Networks (DNN) for phoneme classification [8],[9]. In this work we tried to determine if this approach is useful for audio classification as well. In the description in section 2.3, we show how to use a DNN with the audio data and suggest improvements to the training process.

It is important to note that our DNN classifier was created specifically for the SMART project. Therefore, to show the progress achieved in this project we can only compare it to other classifiers known in the literature. We compare our DNN classifier to two other classifiers. The first one is a Support Vector Machine (SVM) classifier. The second on is based on Gaussian Mixture Models (GMM).

Previous attempts to use DNN with audio data are described in [10] and [11]. In [10] the authors describe music genre classification using convolutional DNN. They report classification rate of about 70% for this task (no comparison to other classifiers is given). In [11] DNN are used to classify audio event during sport events. They achieve 72% classification rate which is somewhat worse than the SVM with 74%.

2.2 Audio data

The original intent of the project was to collect audio samples from outdoor recordings in Santander (D3.4). These samples were intended to be used to train and test the different classification algorithms. Because of privacy concerns the data collection was delayed and still hasn't started at the time of writing of this report (M18). Hence, in order to start the work on the audio classification, we needed to obtain data from other sources.

The audio data for our experiments was obtained from the FreeSound.org site (http://www.freesound.org/), which is a repository of audio samples uploaded by users covering a wide range of acoustic events. From this site we were able to get samples which are either public domain or have a creative commons license (CC-BY).

The sample selection was based on the tags that have been provided by the users. Most of the samples are of outdoor recordings with sounds from ordinary streets and different kinds of live events. A total of 84 files were obtained containing 228 minutes of audio. All samples were uncompressed, converted to one channel and downsampled to 22 KHz if needed.

The samples were then annotated manually. The annotation included locating and labeling segments with the relevant audio classes. Overlapping segments (with two or more audio classes at the same time) were allowed. Table 1 lists the four selected audio classes and the total amount of audio data contained in each class.

Class	**Description**	**Length (minutes)**
Crowd	Crowd of people	63
Traffic	Cars and road noises	37
Applause	Applause, yelling and cheering	27
Music	Various kinds of music recorded outdoors	29

Table 1: The content of the FreeSound data set including the description of the four audio classes and the corresponding amount of audio.

This set of samples presents a harder classification challenge than what we expect from the data that would be collected in the project. The reason for that is that the samples from the site were collected by variety of recording devices and in different environments, whereas in the project the data would come from consistent environments and recording equipment. This increases the variability of the sound patterns and makes it harder for the classifiers to determine the correct classes.

2.3 Audio classification

2.3.1 Audio features

The speech files are divided into 5-second segments with 50% overlap. For each segment we extract a vector of 192 features as described below. The segment itself is divided into frames. Each frame has length of 46 milliseconds and there is a 50% overlap between the frames. The first 64 features are the means of the MFCC parameters over the entire segment. The second set of 64 features is their standard deviation (STD). The last set of 64 features is calculated by first taking the STD of the log spectrum over the frame and then applying the Mel-spaced filter banks.

2.3.2 Classification using DNN

The DNN classifier consists of a multilayer feed-forward perceptron network. Since it is very difficult to train the entire network using discriminative training, a generative pre-training stage is applied first to train one layer at a time. After the pre-training stage, the whole network is fine-tuned using a back propagation process.

The generative training is based on the restricted Boltzmann machine (RBM) model [9]. The RBM is made of one layer network where the input nodes v_i are connected with the binary output nodes $h_j \in \{0,1\}$ using weights matrix W_{ij} and bias vectors b_j and a_i.

The training of the weights is an iterative process. We performed 200 iterations in our experiments. During each iteration, we calculated the hidden layer from the inputs using:

$$P(h_j = 1, v) = \sigma\left(\sum_i W_{ij} v_i + b_j\right), \qquad (1)$$

where

$$\sigma(x) = \frac{1}{1+e^{-x}}. \qquad (2)$$

The inputs are then estimated from the hidden layer using

$$P(\tilde{v}_i = 1, h) = \sigma\left(\sum_j W_{ij} h_j + a_i\right) \qquad (3)$$

for binary inputs and

$$\tilde{v}_i = \sum_j W_{ij} h_j + a_i \qquad (4)$$

for Gaussian inputs. In our case, the inputs for the first layer are Gaussian and binary for the next layers. Finally, the hidden layer is calculated again using the reconstructed inputs from (3) or (4):

$$\tilde{h}_j = \sigma\left(\sum_i W_{ij} \tilde{v}_i + b_j\right). \qquad (5)$$

The weights and bias terms are updated using gradient descent rules. For example, the gradient rule for the weights is based on the correlation between the inputs and the outputs:

$$\Delta W_{ij} = \langle v_i h_j \rangle - \langle \tilde{v}_i \tilde{h}_j \rangle, \qquad (6)$$

where $\langle \cdot \rangle$ represents averaging over the available training data.

The bias terms are updated by:

$$\Delta a_i = \langle v_i \rangle - \langle \tilde{v}_i \rangle \qquad (7)$$

and

$$\Delta b_j = \langle h_j \rangle - \langle \tilde{h}_j \rangle. \qquad (8)$$

If we put aside the original derivation of this process from the RBM formulation, we can consider this as a generative model that tries to minimize the error between v and $\tilde{v}$. Under this perspective we notice that there is a large difference between the binary and the Gaussian cases. Because of (4) the scale of the weights is limited by the scale of the inputs. An attempt to increase the weight to signify a stronger connection would also increase the error between v and $\tilde{v}$. This doesn't happen in the binary case since the estimated values in (3) are always mapped to the region [0,1] using the σ function. Previous attempts tried to solve this by adding the variance into the RBM formulation [12].

In this work we present a novel approach by deviating from the RBM formulation and considering an asymmetric connection between the inputs and outputs. We modify (4) and add scaling factors s_j to the hidden layer nodes. Equation (4) is now converted into:

$$\tilde{v}_i = \sum_j W_{ij} s_j h_j + a_i \,. \qquad (9)$$

The tuning of the new weights is done by taking the gradient direction that minimizes the difference between v and $\tilde{v}$:

$$\Delta s_j = \left\langle h_j \sum_i (v_i - \tilde{v}_i) W_{ij} \right\rangle . \qquad (10)$$

The advantage of this method is that there is no need to change the calculation of the hidden layer in (1) so the same weights are used for the initialization of the neural network.

By applying this technique the pre-training error between v and $\tilde{v}$ for the Gaussian data was reduced by about 19%. We discuss the affect of those weights on the classification results in section 2.4.2.

After the pre-training stage is complete we use the weights and biases ($W_{ij}^{(n)}$ and $b_j^{(n)}$) for each layer n to initialize a feed-forward neural network. We add a final classification layer with one output for each audio class. The network output is calculated as in eq. (1) except that the outputs are not quantized:

$$h_j^{(n)} = \sigma\left(\sum_i W_{ij}^{(n)} h_i^{(n-1)} + b_j \right) , \qquad (11)$$

where $h^{(0)}$ is the input vector and the output of the last layer $h_k^{(N)}$ represents the score for audio class k.

The weights are trained using the neural networks back propagation rule. We are interested in the Equal Error Rate (EER) point but the training data is unbalanced (e.g. 27 minutes of applause data vs. 201 minutes without applause). To fix that we give different weights to the error terms that are used in the calculation of the back propagation gradient. Those weights are inversely proportional to the size of each class.

2.3.3 Classification using SVM

We compared the DNN classifier to an SVM classifier [14]. We used an RBF kernel function (with $\gamma = 0.2$). We trained a one-versus-all binary SVM classifier for each class by taking all the training data of the class as positive examples and all the rest of the data as negative examples. We normalized the input features to the range [0,1] with the scaling estimated from the training data and applied on both the training and the test data. We used the libSVM software (version 3.12) [15] for training and scoring the SVM with probability estimates prediction.

2.3.4 Classification using GMM

Another classifier we used in comparison to the DNN is a simple GMM classifier. We trained a GMM per class using the class data. The classification scores are the log-likelihood score of each class normalized by a score of a Universal Background Model (UBM). The UBM is a GMM model that was trained using all the training data. This approach was successfully used for speaker recognition as described in [16]. However, in our system the UBM and the class models are trained independently without adaptation. We allow each class to have a different number of Gaussians, and optimize the number of Gaussians per class with cross validation. For each class we then train a separate UBM with the same number of Gaussians as in the class model. Because the GMM has diagonal covariance models, we first apply a Principal Component Analysis (PCA) transform to reduce the correlation between the features. We perform the PCA on the input features and select the first 50 components of the transformed feature vector as input for training the GMM. The PCA transformation was trained on the entire training set. We used the Speech Signal Processing Toolkit (SPTK) tool [17] to train and score the GMMs.

2.4 Experiments

We conducted the experiments using the data described in section 2.2. Each classification method was tested 50 times in cross validation using the holdout evaluation method. In each evaluation the speech samples were randomly split into two subsets: 80% of the audio files for training and the rest 20% for testing. Since the holdout evaluation performance depends on the actual split between training and test, we perform random subsampling by repeating each evaluation 50 times and averaging the results. The same data splitting into training and testing subsets were used for the evaluation of all the classification methods.

The classification results were evaluated using the error rate at the equal error rate point (EER). The EER was calculated for each audio class independently and the results were averaged over the 50 Cross Validation (CV) evaluations. In each one of the CV iterations the data set is split randomly to a training set and a test set. We used 80% of the data training and the rest for the test.

We also report the average error over all of the audio classes. In addition to the average EER results, we provide the Standard Error (SE), which is the standard deviation divided by the square-root of the number of evaluations for both the per-class and the overall average results. A large part of the data is labelled with more than one class. Therefore, it is impossible to produce a confusion matrix.

2.4.1 Network structure

We first explored the DNN structure by modifying the number of nodes in the hidden layers. Our DNN has n=3 hidden layers, with the same number of nodes in each layer. The classification results as a function of the number of nodes in the hidden layers is shown in Table 2. As we can see, the class with the lowest EER is the Applause class, and the most difficult class to classify is the Music class. The optimal performance is achieved with 200 nodes. Therefore, in all our following experiments, the DNN will have 3 hidden layers with 200 nodes in each layer. Additional experiments showed that using a deeper network (more hidden layers) provides no additional improvement.

Audio Class	**50**	**100**	**200**	**300**
Crowd	19.67 ± 0.98	18.06 ± 1.11	17.79 ± 0.92	19.81 ± 1.18
Traffic	15.73 ± 0.58	15.53 ± 0.67	15.71 ± 0.67	15.37 ± 0.60
Applause	9.10 ± 1.01	8.8 ± 1.00	8.04 ± 0.85	9.05 ± 1.06
Music	23.78 ± 1.21	23.04 ± 1.23	21.61 ± 1.16	22.15 ± 1.08
Average	**17.07 ± 0.62**	**16.36 ± 0.62**	**15.79 ± 0.57**	**16.60 ± 0.61**

Table 2: Average EER ± SE in percent as a function of the number of nodes in the hidden layers of the DNN

2.4.2 Scaling factors

We have tested what is the impact of the new scaling factors (eq. (9)) by doing the test once with all $s_j = 1$ and then repeating the test and allowing adjustments of the scales by (10). The main impact of the scaling is in reducing the pre-training error between v and $\tilde{v}$ of eq. (10). For the Gaussian data the error was reduced by 19%. This helps to reduce the number of iterations needed by the back-propagation until convergence. This number was reduced by 7%. These results are shown in Table 3.

In terms of the classification performance, the results are displayed in Table 4. As can be seen from this table, the effect of the new technique on the classification results is small (compared to the standard error of the results). By introducing the new scaling factors into the pre-training process a minor degradation in the classification performance is observed. It seems that the back-propagation algorithm is able to achieve similar goals even if the starting point is less optimal.

We expect that this technique will be more useful for reducing the computation time when larger amounts of data are used for the training and larger networks are used for classification [13].

Parameter	**No Scaling**	**Scaling**
RBM error	0.489	0.397
BP iterations	43	40

Table 3: RBM training error after 200 iterations and Back Propagation training iterations without and with the scaling factors in the RBM training of the DNN.

Audio Class	**No Scaling**	**Scaling**
Crowd	17.79 ± 0.92	18.84 ± 1.03
Traffic	15.71 ± 0.67	15.97 ± 0.75
Applause	8.04 ± 0.85	8.76 ± 1.14
Music	21.61 ± 1.16	20.51 ± 1.14
Average	**15.79 ± 0.57**	**16.02 ± 0.60**

Table 4: Average EER ± SE in percent without and with the scaling factors in the RBM training of the DNN.

2.4.3 Comparing with SVM and GMM

In our next experiment, we compare the performance of the DNN to the other two classifiers described in section 2.3. Table 5 shows the classification results of the DNN classifier in comparison with the GMM and SVM classifiers. As we can see, the GMM has worse performance than the other two classifiers, while the SVM and DNN classifiers have comparable performance. The weakness of the GMM classifier for the audio event detection task that we see in our experiment is in line with the findings in [18]. The DNN classifier achieves the best overall performance in most of the per-class results except for the Music class for which the SVM performs better.

Audio Class	**GMM**	**SVM**	**DNN**
Crowd	24.48 ± 0.98	19.82 ± 1.15	17.79 ± 0.92
Traffic	23.85 ± 0.96	16.38 ± 0.77	15.71 ± 0.67
Applause	15.48 ± 1.33	10.18 ± 0.81	8.04 ± 0.85
Music	23.74 ± 1.35	17.77 ± 1.15	21.61 ± 1.16
Average	**21.89 ± 0.64**	**16.04 ± 0.55**	**15.79 ± 0.57**

Table 5: Average EER ± SE performance comparison between three classifiers: GMM, SVM, and DNN.

2.4.4 Score fusion

We perform a simple fusion at the score level by adding the final scores obtained from the DNN and the SVM. The results of the fusion are shown in Table 6. From these results we learn that the score fusion between the DNN and SVM classifiers provides 6.5% improvement relative to the DNN alone. Most of the gain in the fusion is achieved by improvement of the Music classification. Similarly, if we look at the relative improvement of the fused score compared to the SVM, most of the improvement is achieved in the Applause class with overall relative improvement of 8.0%.

Audio Class	DNN+SVM
Crowd	18.04 ± 1.07
Traffic	14.88 ± 0.63
Applause	7.89 ± 0.79
Music	18.21 ± 1.08
Average	**14.76 ± 0.54**

Table 6: Average EER ± SE in percent for score fusion of the DNN and SVM classifiers.

2.4.5 Conclusion

As can be seen from those results, we have shown that we can classify different audio events with good classification rates. It seems that the crowd noises and music are the harder to classify. This is probably since they show the greatest variability (e.g. different types of music).

The main advancement of this work is the successful usage of a DNN classifier. The DNN showed the best performance out of the 3 classifier tested. In addition, we found that additional improvement can be gained by combining the results of two classifiers.

The current classification rate is still not satisfactory and we expect additional improvement. From the small difference in the results of the different classifiers we can conclude that we can't expect a large improvement to be gained by additional improvement to the classifier. It seems that we should now focus on improvement to the features. We expect that we can improve the system by adding different types of features which can better represent the spectral and time dependencies of the different audio events.

2.5 SMART Audio classification modules

Two audio classification modules where created for SMART. The first is a proprietary DNN classifier and the second is an open-source module based on GMM classifiers.

2.5.1 DNN classifier

The above DNN classifier was implemented as a software tool. The goal of this tool is to accept an audio sample as the input, classify short audio segments from this sample and send the results to an EdgeNode server.

This tool is composed of the following modules (see Figure 1):

1. Feature extraction module – receives an audio sample, splits it into segments and calculates the audio features.
2. DNN classifier – uses a pre-trained network to classify the audio features for each segment.
3. EdgeNode interface – provides services for feed registration and sends the classification results to the EdgeNode in JSON format.

The frontend with the feature extraction part is written in C++ for best performance. The backend which contains the other two parts is written in Java.

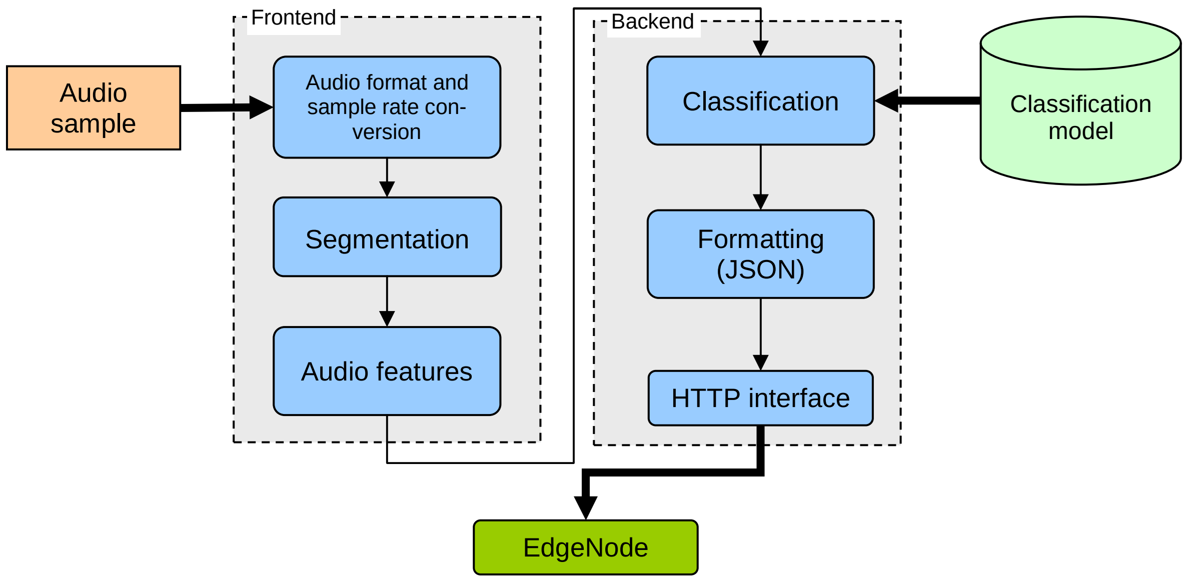


Figure 1: DNN classifier

The tool has two modes of operation:

Initialization

In this mode the tool reads all the configuration data, creates a model file and connects to the Edge-Node to create the feed. Usage:

```
SmartAudioClassifier -i <configuration> <classification model> <classifier file>
```

where:

- **configuration** – input text configuration file
- **classification model** – input text file containing the DNN parameters
- **classifier file** – output classifier file

The format of the configuration and the model files is given at annex 6.1 and 6.2.

Classification

In this mode the tool reads an audio sample, classifies it and sends the results to the EdgeNode. Usage:

```
SmartAudioClassifier <classifier file> <audio file>
```

where:

- **classifier file** – input classifier file that was created in the initialization stage
- **audio file** – input file to be classified

For each audio segment (e.g. 5 seconds) the classifier produces the scores for each of the audio classes. The scores are from 0 to 1 where 1 means high likelihood that audio for this class was present in the segment and 0 means low likelihood.

An example for the JSON output produced by the classifier:

```
{
  "timestamp": 372500,
  "data" : {
    "time": "2013-05-16T17:11:24.134+03:00",
    "audio": {
      "applause_score": 0.005768,
      "music_score": 0.521604,
      "traffic_score": 0.640413,
      "crowd_score": 0.000286,
      "@ID": "audio_classifier_temp",
      "position": 372.500000
    }
  }
}
```

An example for the feed description in XML which also includes the description of the fields in the JSON is given in annex 6.3.

2.5.2 Open source release

A simple implementation of a GMM classifier was released as part of the SMART open source release. We were not expected to deliver any open-source component in the DOW but we decided to do so. The reasons for this release were:

- To increase the potential usage of the open source release of SMART by allowing connection of audio sensors.
- The tool itself is written in Python. By using this very popular scripting language, other developers can easily read the code, modify it to their needs and continue the development of this component.
- The tool provides an interface between the sensors and the EdgeNode. It shows how to register a new feed and how to send the sensor output to the server. Other developers can use this as a hands-on example and reuse the code for connecting other sensors to the server.
- Since the tool uses a simple and common GMM classification algorithm, it can be used as a baseline for evaluating other audio classification algorithms.
- This tool is based on another open source package SPTK[1] which provides the signal processing and GMM classification. This package provides many other signal processing algorithms and tools. This allows other developers to easily experiment with other options such and different features or different classification methods.

This tool adds audio classification abilities to the SMART available sensor inputs.

A detailed description of this tool can be found at on the SMART open source website[2] and in the readme file which comes with the code (also in annex 6.4).

2.6 Plans for next release

The next release is planed at M30. For this release we plan the following improvements:

- Better integration with the SMART sensors (microphones deployed at Santander).
- Improved classification models trained from the data collected at Santander. The new model may be trained specifically for the hardware, environment and sounds. Also, obtaining additional data and annotating it using the video may lead to improvement in the classification

[1] http://sp-tk.sourceforge.net/

[2] http://opensoftware.smartfp7.eu/projects/smart

rates. We may also add more audio classes if these are available in the data.

- Improvements in classification algorithms. As was shown, the differences in the classification rates of the different classification techniques are not very large. We therefore expect that additional improvement can be gained mostly by improving the features used for classification. We are planning to experiment with additional features that can better capture the different audio events.

3 Speech processing

The goal of the speech processing component is to provide a speech interface to the SMART system. This will allow users to provide update messages to the EdgeNode or to the social network manager (SNM) in spoken form.

The speech processing component is described in Figure 2. The process is as follow:

1. A speech sample is recorded by an application. This could be done by any dedicated application running on mobile device.
2. The sample is saved to the speech samples pipeline. This could be done for example by an HTTP servlet which stores the samples on a specific folder or on an FTP site. A text file containing the sample metadata is saved along side the sample.
3. The transcription interface polls the pipeline and pulls a sample from the pipeline when it is available.
4. The sample is sent to the IBM iTrans[3] engine for transcription through its web interface
5. The transcription engine produces the text back to the interface.
6. The result is converted into JSON format and sent to the EdgeNode server.

The output format of the transcription is:

```
{
        "data" : {
                "time" : "<timestamp>",
                "transcription" : {
                        "text" : "<transcription output>",
                        "language" : "<language code (e.g. En-US) used for the transcrip-
tion>",
                        "@ID" : "<component ID>"
                }
        }
}
```

[3] http://researcher.ibm.com/researcher/view_project.php?id=4150

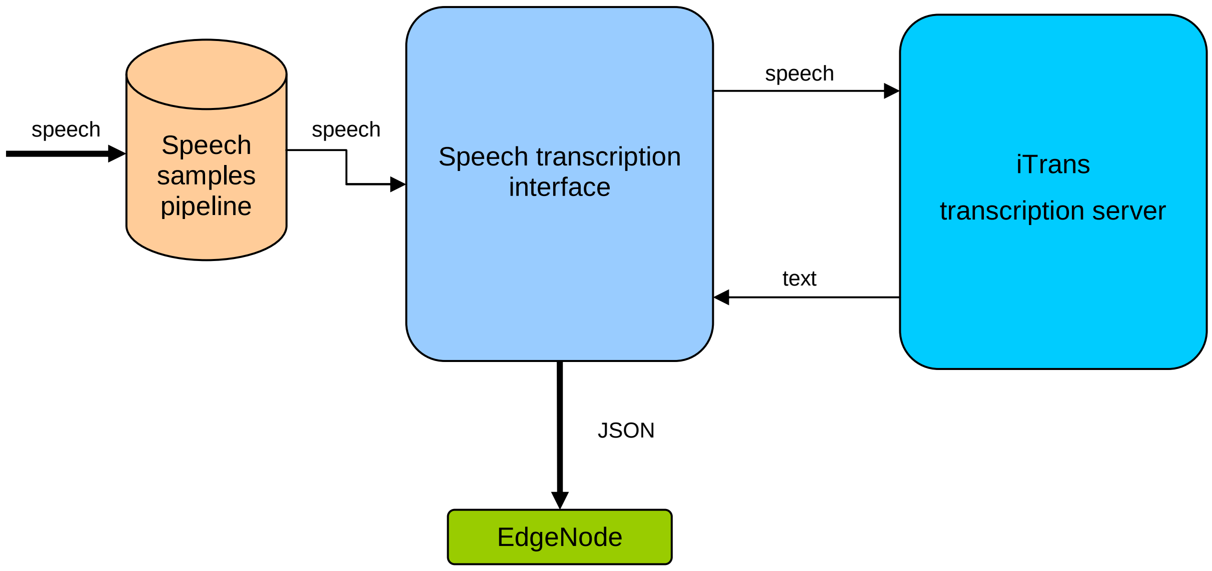


Figure 2: the SMART speech transcription interface

3.1 The iTrans Platform

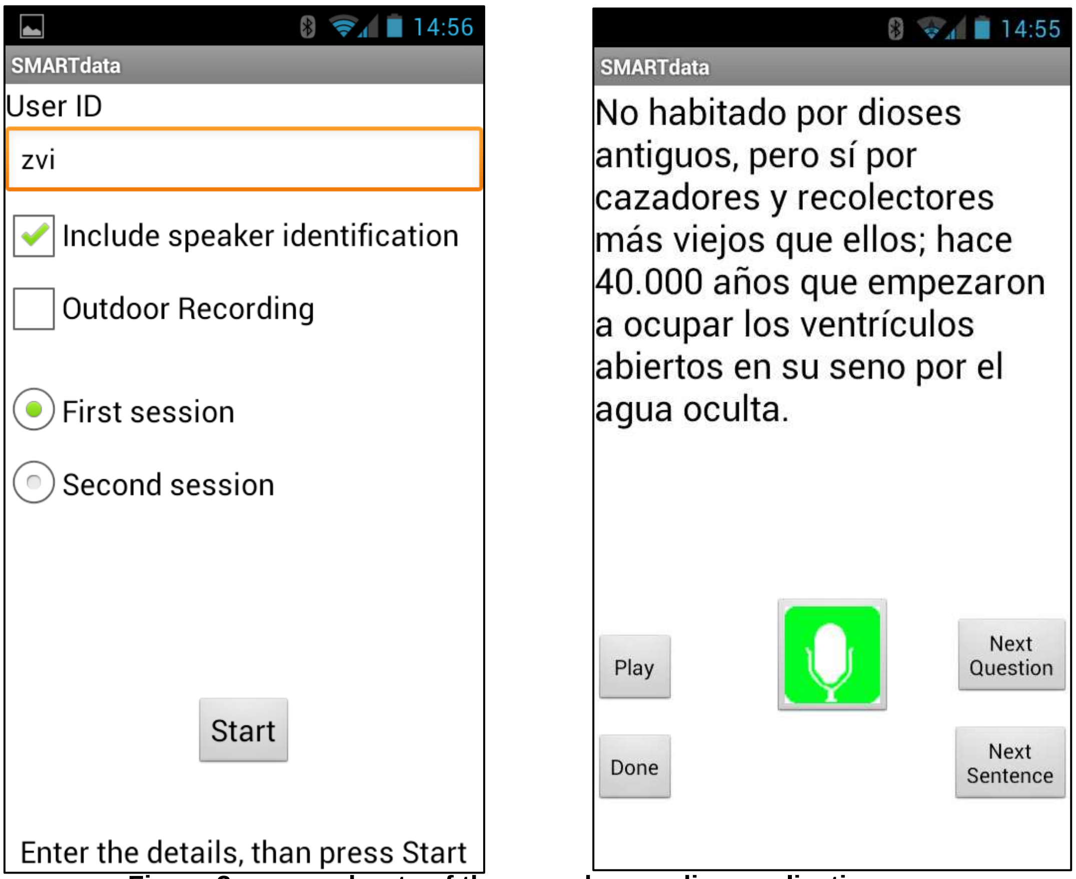
The iTrans is IBM state-of-the art speech transcription middleware. The core Automatic Speech Recognition (ASR) functionality is done using the IBM engine. The iTrans architecture provides a scalable speech transcription cloud service with a set of easy-to-use APIs which make it extremely easy to integrate speech transcription capability into any process. The iTrans platform is currently used in the project without any modification. At later stage a Spanish transcription model will be created for this platform.

3.2 Data collection

The current setting supports only English transcription. The plan is to support also Spanish transcription. In order to support the Spanish language the engine needs to be trained using data in Spanish. This data includes transcribed speech data for training the acoustic models and Spanish text for training the language models.

The collection of this training data (D3.4) was delayed because of the privacy concerns. Recently, after receiving the DPA approval, we have started with the data collection. The collection is done using an application on a mobile phone. The application presents to the user the text to read or questions to answer and then records the speech. Each user records at least one indoor session and one outdoor session and some users also record phrases for speaker verification enrollment. Examples of the application screenshots are presented in Figure 3.

The data collection is done with the help and participants from ATOS, Prisa and Santander.



14:56
SMARTdata
User ID
zvi
[✓] Include speaker identification
[ ] Outdoor Recording
(●) First session
( ) Second session
Enter the details, than press Start

14:55
SMARTdata
No habitado por dioses antiguos, pero sí por cazadores y recolectores más viejos que ellos; hace 40.000 años que empezaron a ocupar los ventrículos abiertos en su seno por el agua oculta.

Figure 3: screenshoots of the speech recording application.

3.3 Spanish transcription model

The Spanish data will be used for training a Spanish transcription model. This model includes two parts: the acoustic model which describes the likelihood of audio data to be related to a phoneme sequence and language model which describe the likelihood of a phonemes sequence to be part of a real sentence. For best performance we'll train both models from data that is related to the domain we're using (e.g. text related to the city of Santander). This is always preferred to using a general model.

The first model is trained using the spoken sentences and the second is trained using large amount of textual data.

3.4 Plans for next release

The next planned release will be on M30. In this release we expect to add the following improvements:

1. Support for Spanish transcription. This will be added after the Spanish models are built using the training data collected.
2. Integration with a speech data collection application.
3. Support for speaker verification. We are currently working on improvements for the speaker verification engine and integration with the transcription server. This will allow identifying specific speakers using a voice signature. This way enrolled user can provide identified message which can have higher priority over messages from the general public. The speaker model will be trained using the collected Spanish data. A full report on the progress in speaker verification would be submitted with the next release.

4 Conclusions

We have presented in this report two prototypes for audio classification and for speech processing. An additional open-source audio classifier was also released outside the scope required by the DOW.

We have shown that the DNN audio classifier we have developed can outperform other classifiers. Additional improvements of the classification could be gained by improving the feature extraction and possibly by additional improvement to the classifier itself.

The main functionality of the speech processing component is currently speech transcription in English. As this is only the first prototype, it is still missing some of the functionality we expect to find in the final version. The additional functionality would include processing of speech in Spanish and the speaker verification component.

5 BIBLIOGRAPHY AND REFERENCES

[1] Aronowitz, H., "Segmental Modeling for Audio Segmentation," *IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. 4, pp.393-396, 15-20 April 2007.

[2] P. Dhanalakshmi, S. Palanivel, and V. Ramalingam. 2009. “Classification of audio signals using SVM and RBFNN”. Expert Syst. Appl. 36, 3 (April 2009), 6069-6075.

[3] McKinney, M. F., Breebaart, J. (2003). “Features for audio and music classification.” Proc. 4th Int. Conf. on Music Information Retrieval.

[4] Zixing Zhang; Schuller, B.; , "Semi-supervised learning helps in sound event classification," *IEEE International Conference on Acoustics, Speech and Signal Processing*, pp.333-336, 25-30 March 2012.

[5] Bach, J.; Meyer, A.; McElfresh, D.; Anemuller, J.; , "Automatic classification of audio data using nonlinear neural response models," *IEEE International Conference on Acoustics, Speech and Signal Processing*, pp.357-360, 25-30 March 2012.

[6] Burred, J.J.; , "Genetic motif discovery applied to audio analysis," *IEEE International Conference on Acoustics, Speech and Signal Processing*, pp.361-364, 25-30 March 2012.

[7] Kumar, A.; Dighe, P.; Singh, R.; Chaudhuri, S.; Raj, B.; , "Audio event detection from acoustic unit occurrence patterns," *IEEE International Conference on Acoustics, Speech and Signal Processing* , pp.489-492, 25-30 March 2012.

[8] Hinton, G. *et al*, “Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups” IEEE Signal Processing Magazine, 2012.

[9] Hinton, G. E., Osindero, S. and Teh, Y., “A fast learning algorithm for deep belief nets.”, Neural Computation, 2006.

[10] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng., “Unsupervised Learning of Hierarchical Representations with Convolutional Deep Belief Networks.” Communications of the ACM, vol. 54, no. 10, pp. 95-103, 2011.

[11] Ballan, Lamberto, et al. "Deep networks for audio event classification in soccer videos." Multimedia and Expo, ICME 2009.

[12] Cho, KyungHyun, Alexander Ilin, and Tapani Raiko. "Improved learning of Gaussian-Bernoulli restricted Boltzmann machines." Artificial Neural Networks and Machine Learning–ICANN 2011 : 10-17.

[13] T. N. Sainath, B. Kingsbury, B. Ramabhadran, P. Fousek, P. Novak and A. Mohamed, "Making Deep Belief Networks Effective for Large Vocabulary Continuous Speech Recognition," in Proc. ASRU, December 2011.

[14] Burges, C. J. C., “A tutorial on support vector machines for pattern recognition,” *Data Mining and Knowledge Discovery*, 2, 121–167, 1998.

[15] Chang, C.-C. and C.-J. Lin. *“LIBSVM: a library for support vector machines”*, 2001. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[16] D. A. Reynolds, T. F. Quatieri ,R. B. Dunn, "Speaker verification using adapted Gaussian mixture models", *Digital Signal Processing*, Vol. 10, No.1-3, 2000.

[17] “Speech Signal Processing Toolkit (SPTK) version 3.6,” Software available at http://sp-tk.sourceforge.net/.

[18] Li Lu,Fengpei Ge,Qingwei Zhao,Yonghong Yan, “A SVM-based Audio Event Detection System”, International Conference on Electrical and Control Engineering, 2010.

6 ANNEXES

6.1 Classifier configuration file

This file has the Java properties file format[4] with the following parameters:

- server = Name of the EdgeNode server (e.g. dusk.ait.gr)
- feed = Name of the feed
- component = Name of the component
- email = address of the user
- description = Text for description of the feed
- tags = list of tags related to the feed
- segment_length = Length of each audio segment (seconds)
- segment_offset = offset in seconds between segments
- frontend = path of the frontend executable
- sample_rate = working sample rate

6.2 DNN file format

The DNN is described by a text file in the following format:

```
AudioClassifier <version number=0>
<NC: number of audio classes>
<name of audio class 1>
<name of audio class 2>
...
<name of audio class NC>
<NF: size of the feature vector>
<mean of feature 1> <STD of feature 1>
...
<mean of feature NF> <STD of feature NF>
<NL: number of layers>
<Layer 1 parameters>
...
<Layer NL parameters>
```

The format of the layers parameters:

```
<N1 N2: layer size>
<N1xN2 layer transfer weights>
<N2 layer bias>
```

[4] http://docs.oracle.com/javase/7/docs/api/java/util/Properties.html#load%28java.io.Reader%29

6.3 Feed description

```
<?xml  version="1.0"  encoding="UTF-8"?><Feed  xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="SMART_Datafeed_Schema_v0.3.xsd">
  <Id>dusk.ait.gr:test_audio_temp_1</Id>
  <Type>Virtual</Type>
  <Title>Audio feed</Title>
  <Description>Audio analysis using Smart DBN audio classifier</Description>
  <DescriptionTags>
    <TextTags>audio</TextTags>
    <URITags>smartfp7:audio</URITags>
  </DescriptionTags>
  <ContactInfo>
    <ContactEmail>zvi@il.ibm.com</ContactEmail>
  </ContactInfo>
  <Components>
    <Virtual>
      <Name>audio_classifier_temp</Name>
      <Description>Example feed description</Description>
      <DescriptionTags>
        <TextTags>ibm.smart.audio.classifier</TextTags>
      </DescriptionTags>
      <Type>audio</Type>
    </Virtual>
  </Components>
  <Outputs>
    <Output>
      <Name>position</Name>
      <ProducedBy>audio_classifier_temp</ProducedBy>
      <Description>Time within the sample</Description>
      <Type>double</Type>
      <Unit>seconds</Unit>
      <HasConfidence>false</HasConfidence>
    </Output>
    <Output>
      <Name>traffic_score</Name>
      <ProducedBy>audio_classifier_temp</ProducedBy>
      <Description>Score for traffic audio</Description>
      <Type>double</Type>
      <HasConfidence>false</HasConfidence>
    </Output>
    <Output>
      <Name>speaker_score</Name>
      <ProducedBy>audio_classifier_temp</ProducedBy>
      <Description>Score for speaker audio</Description>
      <Type>double</Type>
      <HasConfidence>false</HasConfidence>
    </Output>
    <Output>
      <Name>music_score</Name>
      <ProducedBy>audio_classifier_temp</ProducedBy>
      <Description>Score for music audio</Description>
      <Type>double</Type>
      <HasConfidence>false</HasConfidence>
    </Output>
    <Output>
      <Name>applause_score</Name>
      <ProducedBy>audio_classifier_temp</ProducedBy>
      <Description>Score for applause audio</Description>
      <Type>double</Type>
      <HasConfidence>false</HasConfidence>
    </Output>
    <Output>
      <Name>crowd_score</Name>
      <ProducedBy>audio_classifier_temp</ProducedBy>
      <Description>Score for crowd audio</Description>
      <Type>double</Type>
      <HasConfidence>false</HasConfidence>
    </Output>
  </Outputs>
</Feed>
```

6.4 README file for the open source audio classifier

```
SMART Audio classifier
=======================


1. Introduction
================
This package contains sample tools for audio classification. The following functionality
```

```
is provided:

- Training an audio classification model from a set of audio samples.

- Testing the classification rates of the model using a set of audio samples.

- Classifying an audio file and sending the results to an SMART EdgeNode server.

- Creating a feed description file that can be used to register the feed on a SMART Edge-
Node server.


2. Requirements

================

The following packages are required to be installed for this classification tool:

- Python3: http://python.org/ (tested with version 3.2.3)

- Speech Signal Processing Toolkit (SPTK): http://sp-tk.sourceforge.net/ (tested with ver-
sion 3.6)

- SoX - Sound eXchange: http://sox.sourceforge.net/ (tested with version 14.4.0)


3. Configuration

=================

All the tools use the common configuration file: smart/audio/Configuration.py. Before run-
ning any tools it is best to review this configuration file. The user should verify the
"sptk" variable points to the location of the SPTK binaries folder and that the "sox" var-
iable points to the sox binary.


4. Audio data

==============

The classification tools support any audio format which is supported by the sox utility.
During operation the audio files will be converted by "sox" to 16 bit, single channel, raw
PCM file with sample rate specified by the configuration file.


For training and testing the tools require a set of annotated audio files. For each audio
file the user should create a labels files. The label files are plain text files which the
following format:

<t1> <t2> <audio class>

...


where t1 and t2 are the start and end time of the range in seconds and 'audio class' is
one word label of this range. The same classes will be used for classification. The time
ranges can overlap so any given point of time can belong to zero, one or more classes.


An example for a label file could be:

0.0          70.77634604   traffic

92.93723717  146.67739816  traffic

104.15618831 119.39180096  people

130.88776323 145.1538369   people
```

```
141.69119766 203.74169282  music

175.07103992 202.21813156  traffic

206.51180421 237.26004065  traffic


Label files can be created with the help of tools such as Audacity
(http://audacity.sourceforge.net/) and Praat (http://www.fon.hum.uva.nl/praat/).


An audio set is described by a text file containing the paths to pairs of audio and labels
file. An example for audio set file could be:

audio1.wav audio1.lbl

audio2.wav audio2.lbl

...


3. Training audio classification models

=======================================

The train.py is used for training the classification models. For example:


$ export PYTHONPATH="$PYTHONPATH:SmartCode"

$ python3 smart/audio/train.py -m Data/models Data/training_files.txt


This will train the classification modules using the files listed in the train-
ing_files.txt and write the results to the Data/models folder.


Use the '-h' switch for description of all available options. This also works for all the
other commands listed below.


4. Testing classification rates

===============================

The testing.py script is used for testing the classification models. This allows comparing
the labels produced by the models to labels produced manually. It is best to test the mod-
els with files that were not included in the training set. For example:


$ python3 smart/audio/test.py -m Data/models Data/testing_files.txt


This will test the classification modules in the Data/models folder using the files listed
in the testing_files.txt. The output is the equal error rate (EER) for each class.


5. Generating a feed description file

======================================

Feed description files are used to register a feed in an EdgeNode (see
http://opensoftware.smartfp7.eu/projects/smart/wiki/EdgeNodePostCommands#CreateFeed). A
utility is included for generating this file from the audio classes information. This
should be used after the models data is ready.
```

```
For example:

$ python3 smart/audio/description.py -m Data/models -f audio_feed_example -c microphone1 -
  e my_email@address.org -d "Audio classification from outdoor microphone" -t "au-
  dio,street,traffic,people" feed.xml

This will create the feed.xml file. It is recommended to edit this file and verify its
content before posting it to the server.

6. Posting data to the edge server
===================================
The classify utility can process an audio file and post the results to the EdgeNode (see:
http://opensoftware.smartfp7.eu/projects/smart/wiki/EdgeNodePostCommands#Append).

For example:
$ python3 smart/audio/classify.py -m Data/models -e
  http://dusk.ait.gr/couchdb/audio_feed_example -n microphone1 audio.wav

7. License
==========
SMART FP7 - Search engine for MultimediA enviRonment generated contenT
Webpage: http://smartfp7.eu

This Source Code Form is subject to the terms of the Mozilla Public
License, v. 2.0. If a copy of the MPL was not distributed with this
file, You can obtain one at http://mozilla.org/MPL/2.0/.

The Original Code is Copyright (C) 2013 IBM Corp.
All Rights Reserved

Contributor(s):
 Zvi Kons <zvi@il.ibm.com>
```