User’s Guide:
A Noise Classifier Smartphone App
for Hearing Improvement Studies

A. SEHGAL AND N. KEHTARNAVAZ
UNIVERSITY OF TEXAS AT DALLAS

APRIL 2017

This work was supported by the National Institute of the Deafness and Other Communication Disorders (NIDCD) of the National Institutes of Health (NIH) under the award number 1R01DC015430-01. The content is solely the responsibility of the authors and does not necessarily represent the official views of the NIH.
Table of Contents

INTRODUCTION 4
   NOISE CLASSIFIER FOLDER DESCRIPTION 4

PART 1 IOS 6

SECTION 1: IOS GUI 7
   1.1 TITLE VIEW 8
   1.2 SETTINGS VIEW 8
   1.3 STATUS VIEW 9
   1.4 BUTTON VIEW 9

SECTION 2: CODE FLOW 10
   2.1 VIEW 10
   2.2 CONTROLLER 11
   2.3 NATIVE CODE 11

SECTION 3: DATA COLLECTION FOR TRAINING 12
   3.1 READING DATA THROUGH iTUNES (PREFERRED METHOD) 12
   3.2 READING DATA THROUGH XCODE 13
   3.3 DELETING DATA ON APP 14

SECTION 4: TRAINING RANDOM FOREST CLASSIFIER 15

PART 2 ANDROID 16

SECTION 1: ANDROID GUI 17
   1.1 TITLE VIEW 18
   1.2 SETTINGS VIEW 18
   1.3 STATUS VIEW 19
   1.4 BUTTON VIEW 19

SECTION 2: CODE FLOW 20
   2.1 PROJECT ORGANIZATION 20
2.2 Native Code

SECTION 3: DATA COLLECTION FOR TRAINING

SECTION 4: TRAINING RANDOM FOREST CLASSIFIER
Introduction

This user’s guide discusses how to use a noise classifier app that has been developed at the University of Texas at Dallas. Both iOS and Android versions are covered. The noise classifier app uses subband features and a random forest (RF) classifier. The theoretical details of the algorithm and its real-time implementation aspects are published in two papers which can be obtained from the following links:


This guide is divided into two parts. The first part covers the iOS version of the noise classifier app and the second part covers the Android version. Each part consists of 4 sections.

In the first section, the GUI of the app is explained. The second section explains the code flow and how an audio stream is processed to achieve noise classification. Noting that noise classification is done in a supervised manner, the third section shows how the app can be used to collect data for training and the fourth section explains how to use a MATLAB training code to generate the model or the parameters of the random forest classifier.

**Noise Classifier Folder Description**

The code for the Android and iOS versions of the app along with the code for training the RF classifier are provided with this user’s guide. Fig. 1 lists the folder contents of the accompanying code. These contents include:

- “NoiseClassifier_Android” is the Android Studio project. To open the project, open Android Studio, click on “Open an existing Android Studio Project” and navigate to the project “NoiseClassifier_Android”.
- “NoiseClassifier_iOS” is the iOS Xcode project. To open the project, double click on “Noise Classifier iOS.xcodeproj” inside the folder.
- “RandomForest_Training” contains the MATLAB code to train the RF classifier based on data collected by the smartphone on which the app is desired to be run. The folder contains data collected using an iPhone7 and a GalaxyS7 smartphone.
Fig. 1
Part 1

iOS
Section 1: iOS GUI

This section discusses the GUI of the developed noise classifier app and the options included in it, see Fig. 2. The GUI consists of 4 views:

- Title View
- Settings View
- Status View
- Button View
1.1 Title View

The title view displays the title of the app.

1.2 Settings View

This is the view that controls the app settings. It has 6 settings that the user can adjust before the noise classifier app is activated.

- **Sampling Frequency**: The sampling frequency is the frequency at which the audio signal is captured by the smartphone microphone and played by the speaker or via an earphone. iOS mobile devices exhibit the lowest latency at the preferred sampling frequency of 48kHz. The sampling frequency can be adjusted with the help of a slider to the following frequencies in kHz: 8, 11.025, 12, 16, 22.05, 24, 32, 44.1, and 48.

- **Window Size**: This indicates the amount of data samples in milliseconds that get processed by the noise classifier app. The processing frame denotes the number of samples corresponding to the window size. The following equation is used to translate time into number of samples:

  \[
  Window\ Size\ (in\ samples) = \frac{Window\ Size\ (in\ ms) \times Sampling\ Frequency\ (in\ Hz)}{1000}
  \]

  Audio frames are processed with 50% overlap. This means that the content of a processing frame is updated at half the window size time. This is done using a Hanning window on a processing frame to minimize the effect of signal leakage.

- **Decision Rate**: Decisions made by the classifier are smoothed out using a majority voting buffer. Decisions are collected over a decision time duration and then the most popular decision during this time is used as the detected noise class. This is done to remove transient misclassifications.

- **Quiet Adjustment**: For situations when the noise power is low and no noise reduction would be necessary, a quiet condition is included. This is a simple signal power threshold in dBFS (dB Full Scale). A nominal threshold value is included which can be adjusted by the user by using the button +/-.

  If the noise becomes bothersome and the condition shows quiet, the user can decrease the threshold. If the noise is not bothersome, the user can increase the threshold.

- **Play Audio**: This switch enables or disables the audio output to the speaker.

- **Store Features**: Enabling this switch allows the app to collect features of noise signals from a noise environment. This allows collecting data for training the classifier using the smartphone on which the app will be running.
1.3 Status View

The status view provides feedback to the user, indicating a currently detected noise environment and how much processing time is taken per frame. The displayed frame processing time is averaged over 0.5 seconds.

1.4 Button View

This view allows one to start or stop the app. Note that the app can operate on a file as well. The stop button is enabled when the app gets started. The settings cannot be changed after starting the app.

To include an audio file to be read by the app, see Fig. 3, add the audio file to the project. Make sure that the audio file is in “.wav” format.
Section 2: Code Flow

This section discusses the code flow of the app. The app is designed to make the components modular and the code blocks can easily be replaced. One can view the code by running “Noise Classifier iOS.xcodeproj” in the folder “NoiseClassifier_iOS” as shown in Fig. 1. The code is divided into 3 sections as shown in the project navigator in Fig. 4:

- View
- Controller
- Native Code

2.1 View

The app view is done through its GUI. The initial setup of the GUI along with the screen interaction methods, e.g. button touch, slider adjust, etc., are mentioned here. The GUI consists of 3 parts:

- **Main.storyboard**: This provides the layout of the GUI.
- **ViewController**: This connects the GUI elements with actions.
- **popoverViewController**: This is the view associated with the button “Read File”. It populates a table view where the user can choose the file to be read.

### 2.2 Controller

Through the controller, data are passed to the native code, the status from the native code appears in the GUI, and the app I/O is initialized. The controller consists of 2 parts:

- **GlobalVariable**: This acts as a data bridge between the native code and the view. It allows the data entered by the user to be accessed by the native code. Also, it reports the running status of the app back to the user through the user interface.
- **iosAudioController**: This controls the audio i/o setup and calls the native code to process each incoming audio frame.

### 2.3 Native Code

The native code comprises the engine of the app. It processes each incoming frame and outputs the detected noise type. It is divided into the following parts:

- **SpeechProcessing**: This is the main component of the native code. It initializes all the settings and then assigns how each module should be called in order to classify the noise environment. The following modules are used for the noise classification:
  - **Transforms**: This module computes the incoming FFT of audio frames.
  - **SubbandFeatures**: This module uses the FFT to extract the subband features. The number of bands to be used can be set in the code. It is either 4 or 8.
  - **RandomForest**: This module classifies the noise environment based on the subband features.
  - **TrainData**: This is the random forest classifier designed to carry out the classification. It is obtained by performing training in MATLAB on actual collected data.

The above modules are written in a modular manner allowing one to add and replace audio processing, and to perform feature extraction and noise classification tasks with ease.
Section 3: Data Collection for Training

Considering that the classifier is a supervised classifier, data are needed to train it. This section shows how to collect data using the app, and how to train the RF classifier based on the collected data. Data can be collected using the “Store Features” switch button in the app.

When this button is turned on and the app is run, the app stores the data needed for training the classifier. The data get stored in the form of a “.txt” file with comma separated values and each data frame appears on a new line.

Below is an example of how the data get stored when the number of subband features is considered to be 4.

<table>
<thead>
<tr>
<th>BP1</th>
<th>BP2</th>
<th>BP3</th>
<th>BP4</th>
<th>BE1</th>
<th>BE2</th>
<th>BE3</th>
<th>BE4</th>
<th>MB OP</th>
<th>RF OP</th>
<th>dB Power</th>
</tr>
</thead>
</table>

BP – Band Periodicity
BE – Band Entropy
RF – Random Forest
MB – Majority Voting Buffer
OP – Output
dB Power (in dBFS)

The data get stored for every frame on a new line. The data do not get stored when no classification is made or the environment is labelled as quiet.

The data that get stored can be read in two ways:

3.1 Reading Data through iTunes (Preferred Method)

- Connect the smartphone to a Mac computer
- Open iTunes
- Open the status page of the smartphone
- Click on Apps on the sidebar
- Scroll to the bottom to the “File Sharing” section
- Click on “Noise Classifier iOS”; one should be able to see the recorded data files. The file naming format is “MM-DD-YY-HH-MM-SS.txt”
- Select the desired files to save and then click the button “Save to...” as shown in Fig. 5
3.2 Reading Data through Xcode

The data can also be read from the app project in Xcode as follows:

- Connect the smartphone to Xcode
- Open the file “AppDelegate.m” through the project navigator
- Uncomment the line “#define READ_DATA” and run the app
- The data will be displayed in the console Xcode as shown in Fig. 6
3.3 Deleting Data on App

When the app is uninstalled, all the data collected will be deleted. This is the **safe way** to delete the collected data.

The other way is through the file “AppDelegate.m”. Uncomment the line “#define DELETE_DATA” in the code and run the code. All the stored files will get deleted.

**Warning:** Remember to comment the line “#define DELETE_DATA” before running the application again. Otherwise, every time the application is loaded, it will delete the previously stored data.
Section 4: Training Random Forest Classifier

After data are collected, the RF classifier needs to be trained for its model or parameters to get generated in MATLAB. The generated model is then assigned to the native code for the classification operation.

Here, two noise classes are considered: Stationary and Non-Stationary. The data collected for these two classes are stored in the folder “Data” located inside the folder “RandomForest_Training”. The folders containing the files are labelled as “Class_Non-Stationary” and “Class_Stationary”. The labels generated for the classes will be the names of the folders with “Class_” removed.

After data are stored in the folders, execute the file “Initialize_random_forest_training.m” when running the random forest training for the very first time. After that, run the script “Train_Random_Forest.m” and the model gets generated as a header file stored in the folder “Random_Forest_Model”.

Fig. 7 shows how the folder structure appears for training. Also, note that the random forest model header is generated with the date and time in the file name in order to keep track of when it was generated. To implement the model, simply copy the contents of the header file to the file “TrainData.h” in the Xcode project.
Part 2
Android
Section 1: Android GUI

This section discusses the GUI of the Android version of the app and the features of the GUI, see Fig. 8. The GUI is divided into 4 views:

- Title View
- Settings View
- Status View
- Button View
1.1 Title View

The title view displays the title of the app.

1.2 Settings View

This is the view that controls the app settings. It has 6 settings that the user can adjust before the noise classifier app is activated.

- **Sampling Frequency**: The sampling frequency is the frequency at which the audio signal is captured by the smartphone microphone and played by the speaker or via an earphone. Android mobile devices exhibit the lowest latency at the preferred sampling frequency of 48kHz. The sampling frequency can be adjusted with the help of a slider to the following frequencies in kHz: 8, 11.025, 12, 16, 22.05, 24, 32, 44.1, and 48.

- **Window Size**: This indicates the amount of data samples in milliseconds that get processed by the noise classifier app. The processing frame denotes the number of samples corresponding to the window size. The following equation is used to translate time into number of samples:

  \[
  \text{Window Size (in samples)} = \frac{\text{Window Size (in ms)} \times \text{Sampling Frequency (in Hz)}}{1000}
  \]

  Audio frames are processed with 50% overlap. This means that the content of a processing frame is updated at half the window size time. This is done using a Hanning window on the processing frame to minimize the effect of signal leakage.

- **Decision Rate**: Decisions made by the classifier are smoothed out using a majority voting buffer. Decisions are collected over a decision time duration and then the most popular decision during this time is used as the detected noise class. This is done to remove transient misclassifications.

- **Quiet Adjustment**: For situations when the noise power is low and no noise reduction would be necessary, a quiet condition is included. This is a simple signal power threshold in dBFS (dB Full Scale). A nominal threshold value is included which can be adjusted by the user by using the slider. If the noise becomes bothersome and the condition shows quiet, the user can decrease the threshold. If the noise is not bothersome, the user can increase the threshold.

- **Play Audio**: This switch enables or disables the audio output to the speaker.

- **Store Features**: Enabling this switch allows the app to collect features of noise signals from an environment. This allows collecting data for training the classifier using the smartphone on which the app will be running.
1.3 Status View

The status view provides feedback on a real-time basis, indicating a currently detected noise environment and how much processing time is taken per data frame. The displayed frame processing time is averaged over 0.5 seconds.

1.4 Button View

This view allows one to start or stop the app. Note that the app can operate on a file as well. The stop button is enabled when the app gets started. The settings cannot be changed after starting the app.

To read files, store them on the device in the folder labelled “NCAndroid”. When one clicks on the button “Read File”, a list will pop up with the stored audio files. Make sure that the files stored are in “.wav” format.
Section 2: Code Flow

To be able to run the app, it is necessary to have the Superpowered SDK on the smartphone device. This SDK can be acquired from the following link:

http://superpowered.com/

The Android shell can be opened in Android Studio using the following steps:

- Start Android Studio
- Select “Open an existing project”
- Navigate to the folder “NoiseClassifier_Android”
- Click open

After the project is opened, navigate to the file “local.properties” in the project. In the project, make sure that “sdk.dir” (Android SDK), “ndk.dir” (Android NDK) and “superpowered.dir” (Superpowered Audio Engine) point to the proper location. Clean your project and rebuild it so that the project can be run on the smartphone device.

2.1 Project Organization

The project is organized in the following way in Android Studio which can be seen under the view “Android”, see Fig. 9, in the project navigator as shown in Fig. 10:

- **java**: This folder contains the file “MainActivity.java”. This file handles all the operations of the app and allows one to link the GUI to the native code.
- **cpp**: This folder contains the native code. This folder has two subfolders.
  - **AndroidIO**: This subfolder contains the Superpowered source root files which control the audio interface to the app.
  - **jni**: This subfolder includes all the native code required to start the audio i/o and also the noise classifier files. This code provides the audio processing part of the app.

2.2 Native Code

The native code is the engine of the app. It processes each incoming frame and outputs the detected noise type. It is divided into the following parts:

- **FrequencyDomain**: This is the C++ file that acts as a bridge between the native code and the java GUI. This file is responsible for creating the audio i/o interface with the settings
appearing in the GUI, for providing the audio data to the processing code frame-by-frame and for reporting the result back to the GUI.
• **SpeechProcessing**: This is the main component of the native code. It initializes all the settings and then assigns how each module should be called in order to classify the noise environment. The following modules are used for the noise classification:
  o **Transforms**: This module computes the incoming FFT of the audio frame.
  o **SubbandFeatures**: This module uses the computed FFT to extract the subband features. The number of bands to be used can be set in the code. It is either 4 or 8.
  o **RandomForest**: This module classifies the noise environment based on the computed subband features.
  o **TrainData**: This is the model of the random forest classifier used for the noise classification. It is obtained by performing training on real data in MATLAB.

The above modules are written in a modular manner allowing one to add and replace audio processing, and to perform feature extraction and noise classification tasks with ease.
Section 3: Data Collection for Training

Considering that the classifier is a supervised classifier, data are needed to train it. This section shows how to collect data using the app, and how to train the RF classifier based on the collected data. Data can be collected using the switch button “Store Features” in the app.

When this button is turned on and the app is run, the app stores the data needed for training the classifier. Data get stored in the form of a “.txt” file with comma separated values and each data frame appears on a new line.

Below is an example of how the data get stored when the number of subband features is considered to be 4.

<table>
<thead>
<tr>
<th>BP1</th>
<th>BP2</th>
<th>BP3</th>
<th>BP4</th>
<th>BE1</th>
<th>BE2</th>
<th>BE3</th>
<th>BE4</th>
<th>MB OP</th>
<th>RF OP</th>
<th>dB Power</th>
</tr>
</thead>
</table>

BP – Band Periodicity
BE – Band Entropy
RF – Random Forest
MB – Majority Voting Buffer
OP – Output
dB Power (in dBFS)

The data get stored for every frame on a new line. The data do not get stored when no classification is made or the environment is labelled as quiet. The data are stored as text files on the smartphone device in the device storage under the folder labelled “NCAndroid”. The file name is kept as “yyyy-MM-dd-HH-mm-ss.txt” to keep track of when the features were stored. The files can be easily accessed and deleted if needed on the smartphone used.
Section 4: Training Random Forest Classifier

After data are collected, the RF classifier needs to be trained for its model or parameters to get generated in MATLAB. The generated model is then assigned to the native code for the classification operation.

Here, two noise classes are considered: Stationary and Non-Stationary. The data collected for these two classes are stored in the folder “Data” located inside the folder “RandomForest_Training”. The folders containing the files are labelled as “Class Non-Stationary” and “Class Stationary”. The labels generated for the classes will be the names of the folders with “Class_” removed.

After data are stored in the folders, execute the file “Initialize_random_forest_training.m” when running the random forest training for the very first time. After that, run the script “Train_Random_Forest.m” and the model gets generated as a header file stored in the folder “Random_Forest_Model”.

Fig. 11 shows how the folder structure appears for the training folder. Also, note that the random forest model header is generated with the date and time in the file name to keep track of when it was generated. To implement the model, simply copy the contents of the header file to the file “TrainData.h” in the project.