IMPLEMENTATION OF A SPOKEN 4-FUNCTION CALCULATOR IN MATLAB

Jordan Floyd

Department of Electrical and Computer Engineering
University of Rochester

ABSTRACT

There is a growing prevalence of voice assistants in the modern world. As people rely more on voice assistants to help with everyday tasks, the need for better speech recognition software becomes greater. This makes it worthwhile to investigate the different methods used for speech recognition.

In this paper, an application of MFCCs (mel-frequency cepstral coefficients) for speech-recognition purposes is described and evaluated. The MFCCs of thousands of audio files are extracted in order to train two models. The models are then used to determine the contents of a spoken mathematical expression. The program evaluates that expression and returns the full equation to the user.

Although there is plenty of room for improvement, this project shows that even simple models can be somewhat successful in distinguishing between more complex words. Perhaps a more valuable outcome of this project was the learning experience that came along with completing it.

Index Terms— machine learning for audio, MFCCs, audio signal processing, ECOCs, speech recognition

1. INTRODUCTION

As voice assistants increase in popularity, so will the demand for better accuracy and improved functionality of the programs. Although a spoken 4-function calculator has limited capabilities, the implementation process can give an individual valuable intuition and experience. This can help them complete more complex and practical projects in the future.

The interactions between the spoken 4-function calculator and a user is similar to those of voice assistants. Similar to a wake word, the user presses the "Run" button in MATLAB to signal that they would like to use the calculator. Once the word "RECORDING" appears in the command prompt, the user should say a simple mathematical expression. After 5 seconds have passed, the words "DONE RECORDING" appear in the command prompt. Like a voice assistant, the program now needs to process the input audio signal. Finally, it displays the expression that it thought the user said along with the result of that expression. A typical voice assistant would do this verbally.

2. BACKGROUND

2.1. Mel-Frequency Cepstral Coefficients (MFCCs)

MFCCs are "the most popular spectral based parameter used in [speech] recognition" tasks [1]. This makes sense because the mel-frequency scale is based on human auditory perception. A speech signal with a linear frequency f_{lin} can be mapped to the perceived (mel-scale) frequency f_{mel} by using the following equation [1].

$$f_{mel} = 2595 * log_{10}(1 + \frac{f_{lin}}{700})$$

2.2. Error-Correcting Output Codes (ECOCs)

An ECOC model reduces a multiclass classification problem to "a set of binary classification problems" like those solved by support vector machines (SVMs) [2]. The "fitcecoc" MATLAB function allows an individual to train an ECOC model.

3. METHODS & RESULTS

3.1. Spoken-Digit and Spoken-Function Datasets

Two datasets are necessary in order to train models to recognize spoken digits and spoken functions. A free dataset of spoken digits is available on GitHub [3]. However, I was unable to find a spoken-function dataset. Accordingly, a new dataset had to be created to accomplish the goals of this project.

The spoken-digits dataset contains a total of 2000 recordings from 4 speakers (50 recordings per digit per speaker). Once the files were downloaded from GitHub, they were organized into "train" and "test" sets as well as into folders labeled with the number that is being spoken. The first 5 recordings of each digit from each speaker went to the "test" set. Therefore, the "test" set makes up 10% of the total dataset.

The spoken-function dataset consists of roughly 800 recordings from 4 speakers (about 50 recordings per function per speaker - some speakers said some words a few times more or less than what was requested). These recordings were made in Audacity. After a speaker finished saying a word 50 times, each repetition of the word was labeled one-by-one. Then, the audio snippets associated with the labels were exported to separate .wav files by using the "Export Multiple" option. These .wav files were saved to the appropriately-labeled folder within either "test" or "train" sets. Similar to the spoken-digit dataset, the "test" set makes up about 10% of the total dataset. Figure 1 illustrates how the audio snippets were labeled in Audacity.

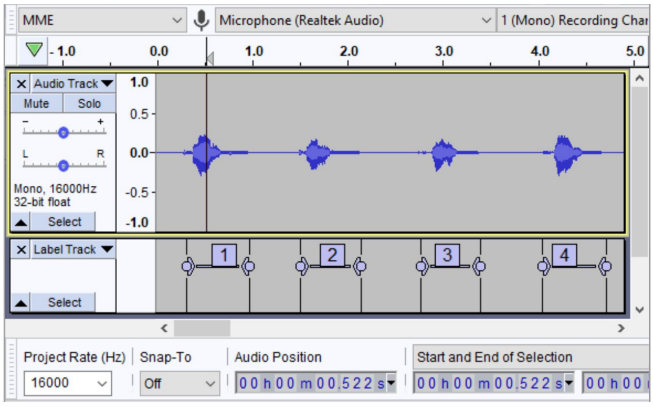

Figure 1: Labeling Separate Snippets of a Spoken Function ("Plus") in Audacity

3.2. Model Training and Testing

The Audio Signal Processing course at the University of Rochester includes an assignment on Machine Learning for Audio. Along with the assignment, students are given a MATLAB function ("my_mfcc.m") that extracts MFCC features, a MATLAB data file ("melbanks.mat") that contains mel filter banks, and some skeleton files. Several of these files were used to implement the spoken 4-function calculator.

To train the spoken-digit model, I wrote a "for" loop to access each audio file in each folder of the "train" set. Since all of the audio signals in this dataset have a sampling frequency of 8 kHz and the "my_mfcc.m" function requires a 16 kHz sampling frequency, I had to use the "resample" MATLAB function to resample the audio signals to 16 kHz.

To increase the accuracy of the model, I clipped or zero-padded each audio signal to a consistent 4500 samples. A slight problem with this is that the exact content of each signal is no longer the same due to the varying speeds of the speakers.

Next, I extracted the MFCCs of each recording and appended them to an array called "x_train_digits." I also appended a set of labels corresponding to the content of the audio file to another array called "t_train_digits." Once these arrays were complete, I used the built-in MATLAB function "fitcecoc" to train the spoken-digit model.

To test the model, I followed the same process as I did in "p3_test.m" of the Machine Learning for Audio assignment. I used "my_evaluation_digits.m" (based on "my_evaluation.m" from the assignment) to print out a confusion matrix (shown in Figure 2) so I could evaluate my results.

```
%%%%%%%% confusion matrix %%%%%%%%
          0       1       2       3       4       5       6       7       8       9
0      47.88%  1.92%   0.58%  15.38%  5.00%   3.27%   6.54%   0.19%  16.35%  2.88%
1      10.58%  51.73%  13.85%  0.77%   0.00%  11.15%  1.73%   0.96%   0.58%   8.65%
2      1.73%   16.35%  30.38%  1.15%   0.19%   9.62%   4.04%  13.85%  2.12%  20.58%
3      7.50%   0.19%   1.35%  82.12%  2.69%   1.35%   2.50%   0.19%   1.92%   0.19%
4      0.96%   0.38%   0.96%   2.12%  73.08%  4.42%  12.12%   0.77%   4.81%   0.38%
5      7.31%  14.04%   5.19%   0.58%   0.77%  49.04%  7.50%  10.19%   0.77%   4.62%
6      1.92%   1.54%   2.88%   0.00%   7.12%  13.85%  63.27%   0.00%   7.88%   1.54%
7      1.92%   7.31%   4.81%   0.58%   2.69%  17.50%  2.69%  56.92%   2.12%   3.46%
8      9.23%   0.38%   2.12%   0.00%   3.46%   4.04%   4.62%   0.96%  69.62%  5.58%
9      3.46%   7.88%  20.96%   7.12%   0.38%   3.27%   2.31%   2.50%   1.92%  50.19%
```

Figure 2: Confusion Matrix for the Spoken-Digit Test Set

The training and testing scripts of the spoken-function model are very similar to those of the spoken-digit model. However, the audio signals did not need to be resampled, and I clipped or zero-padded each audio signal to a consistent 6000 samples. The confusion matrix for the spoken-function model is shown in Figure 3.

```
%%%%%%%% confusion matrix %%%%%%%%
          +       -       *       /
+      75.28%  7.08%   7.50%  10.14%
-      2.27%   77.40%  13.26%  7.07%
*      21.69%  16.27%  51.72%  10.32%
/      13.47%  18.33%  10.83%  57.36%
```

Figure 3: Confusion Matrix for the Spoken-Function Test Set

3.3. Implementation of Spoken Calculator

To record input audio for 5 seconds, an "audiorecorder" object is created in MATLAB, and its "recordblocking" method is used. Once the audio data is retrieved with the

"getaudiodata" method, the signal is divided into 3 segments programmatically. If the user used the program as I intended, these 3 segments should correspond to the first digit, the function, and the second digit, respectively. Figure 4 provides a visual example of the 3 segments.

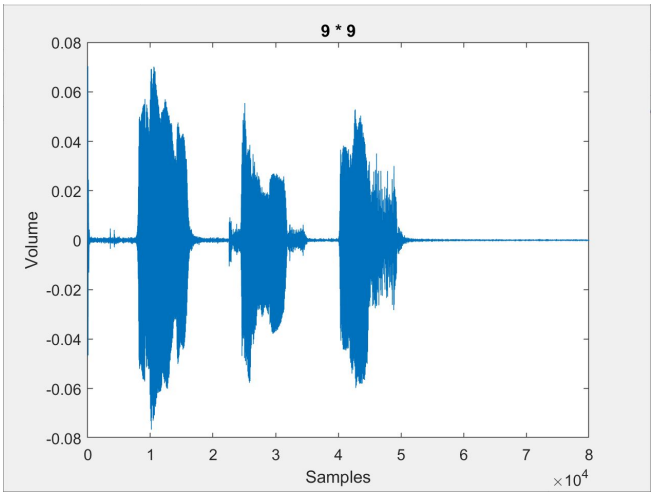

Figure 4: Plot Showing 3 Separate Segments of Input Audio Signal (Spoken Words "Nine Times Nine")

For each of the three segments, the "predict" MATLAB function is used to classify each segment into either a digit (segments 1 and 3) or a function (segment 2). The "predict" function returns a vector of labels. These labels correspond to the model's predictions of the digit/function. The program takes the mode of this vector in order to classify the digit/function. Finally, a "switch" statement is used to print out the predicted expression and its answer in the command prompt. Figure 5 shows the correct output of the program if the user says "nine times nine."

```
>> spoken_calculator_v2
Please wait until you see the word "RECORDING."
Please speak clearly and somewhat slowly.
RECORDING.
DONE RECORDING.
9 * 9 = 81
```

Figure 5: Sample Output of Spoken 4-Function Calculator

4. CONCLUSIONS & FUTURE WORK

4.1. Improving Accuracy

As one can see in Figures 2 and 3, the accuracy of the models leaves room for improvement. A straightforward way to improve the accuracy is to add more recordings to the datasets. As the speaker quantity and speaker variety increase, the models become more representative of the general population.

As audio becomes longer and more complex, it becomes more difficult to classify with a function like "fitcecoc." If utilizing a more complicated method is out of the question, a different approach to model-training may yield better results. It would be interesting to see what would happen if models were trained based on a distinct phoneme of each of the digits/functions instead.

4.2. Expanding on Functionality

While there are countless ways in which one can expand on the capabilities of the spoken 4-function calculator, I have included some ideas that I believe are among the most relevant and useful.

- To make the spoken 4-function calculator entirely hands-free, a wake word could be implemented.
- The calculator could be made more practical by allowing operations between larger numbers.
- Having the calculator respond verbally as well as visually could improve the accessibility and convenience of the calculator.

5. REFERENCES

[1] Ittichaichareon, Chadawan, Siwat Suksri, and Thaweesak Yingthawornsuk. "Speech recognition using MFCC." In International Conference on Computer Graphics, Simulation and Modeling (ICGSM'2012), pp. 28-29. 2012.

[2] "ClassificationECOC Documentation." MathWorks. 2020. https://www.mathworks.com/help/stats/classificationecoc.html

[3] Jakobovski. "Free Spoken Digit Dataset." GitHub. 2019. https://github.com/Jakobovski/free-spoken-digit-dataset

[4] Dietterich, Thomas G., and Ghulum Bakiri. "Error-correcting output codes: A general method for improving multiclass inductive learning programs." In AAAI, pp. 572-577. 1991.