# Laboratory 2 – Classification using Frequency Domain Features

## Objective

This lab has two parts. The first involves decoding phone numbers from audio files recording touchtone dialing. For the second part, you will classify sound files to distinguish between the words "yes" and "no."

## Introduction

Touchtone phones were introduced around 1963, and eventually replaced pulse dial rotary phones by the 1990's. Believe-it-or-not, you could not own a telephone in the United States until 1983 (at least one that you could use with the phone company). You had to lease your phone from AT&T (Bell Telephone). When touchtone phones were introduced, you had to pay an extra fee to use the touchtone service[1]. A major advantage of touchtone phones is that you can enter phone numbers faster. With a rotary phone, you had to stick your finger in the hole in the dial at the number you were entering, turn the dial until it stopped, and then wait for the dial to return to its initial position, which took some time. Also, twisting the dial around, took a toll on your fingers. In addition to faster dialing, the touchtone system is more accurate, enables additional features, and is easier on the fingers. For this lab, you will decode phone numbers by Fourier analysis of audio recordings of touchtone dialing.

The original 1960's Star Trek series amazed viewers with Captain Kirk speaking to his computer, and it understanding him! At the time, I wondered if this incredible technical leap forward could really be achieved in only 200 years? Speech recognition has a long and storied history. Automated speech recognition dates back to the 1950's when Bell Labs developed the Audrey system that could recognize digits spoken by a single speaker. In the 1970's, Carnegie Mellon developed the Harpy speech system that could recognize about 1,000 words, but was still speaker-dependent. Around this time, Bell Labs improved their system to interpret multiple speakers. In the 1980's, people began using Hidden Markov Models to predict the probability of spoken words. In the 1990's, Dragon Dictate was introduced which, when trained on a single speaker, could transcribe spoken words with reasonable accuracy. BellSouth developed the voice portal system for dial-in interactive voice recognition used in the ever-popular phone tree systems that still frustrate us today. By the 2000's speech recognition had reached about an 80% accuracy using frequency-domain features and traditional probabilistic methods. More recently, artificial intelligence and deep learning have accelerated progress in speech recognition with systems like Alexa and Google Assistant achieving accuracies of around 95%. A key concept necessary for speech recognition systems is the analysis of sound in the frequency domain. That is what you will investigate in a simple exercise to distinguish two words, "yes" and "no."

[1] Bell Telephone had a lot of ways to get more money from their customers. If you wanted a color telephone instead of the standard black handset, you paid extra! There are probably still people who continue to lease their phones from AT&T today. There was an article a few years ago about an 82 year old woman who kept the lease on her black rotary phone from 1960, paying $29.10 a month. Her grandchildren discovered that she had spent over $14,000 leasing her phone.

**Instructions**

Part A – Phone number recognition from touch tones.

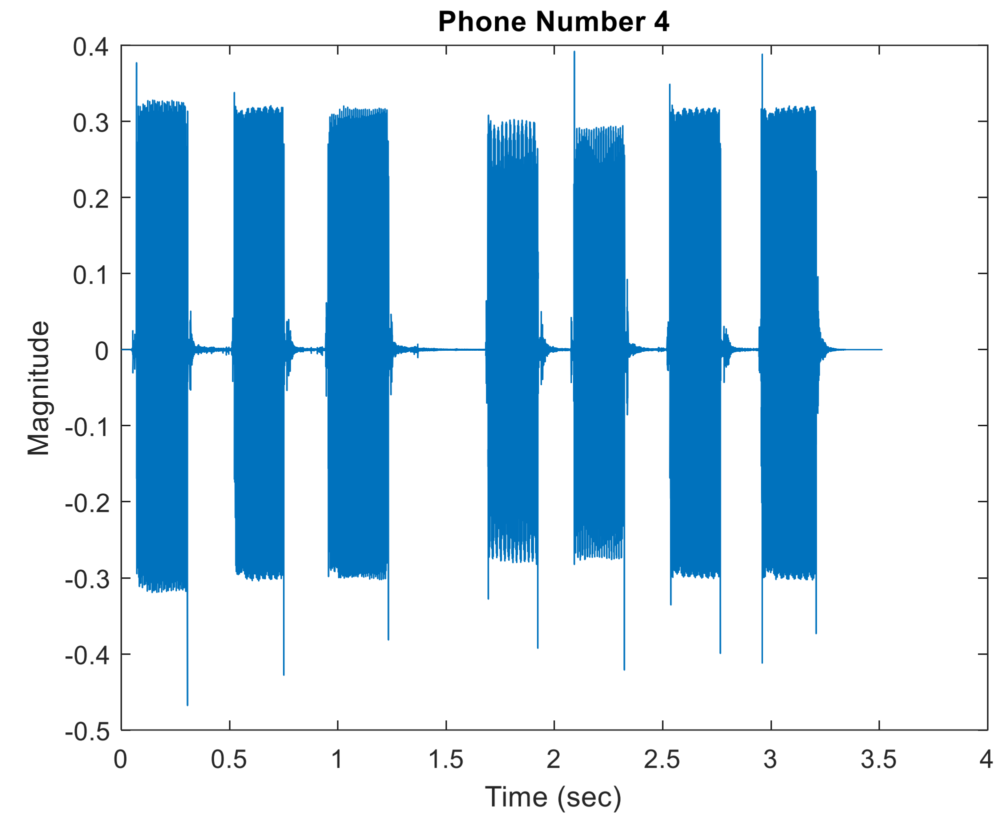
A directory with .mp3 recordings of nine phone numbers can be found on the course Moodle site at https://moodle.swarthmore.edu/mod/folder/view.php?id=730718.

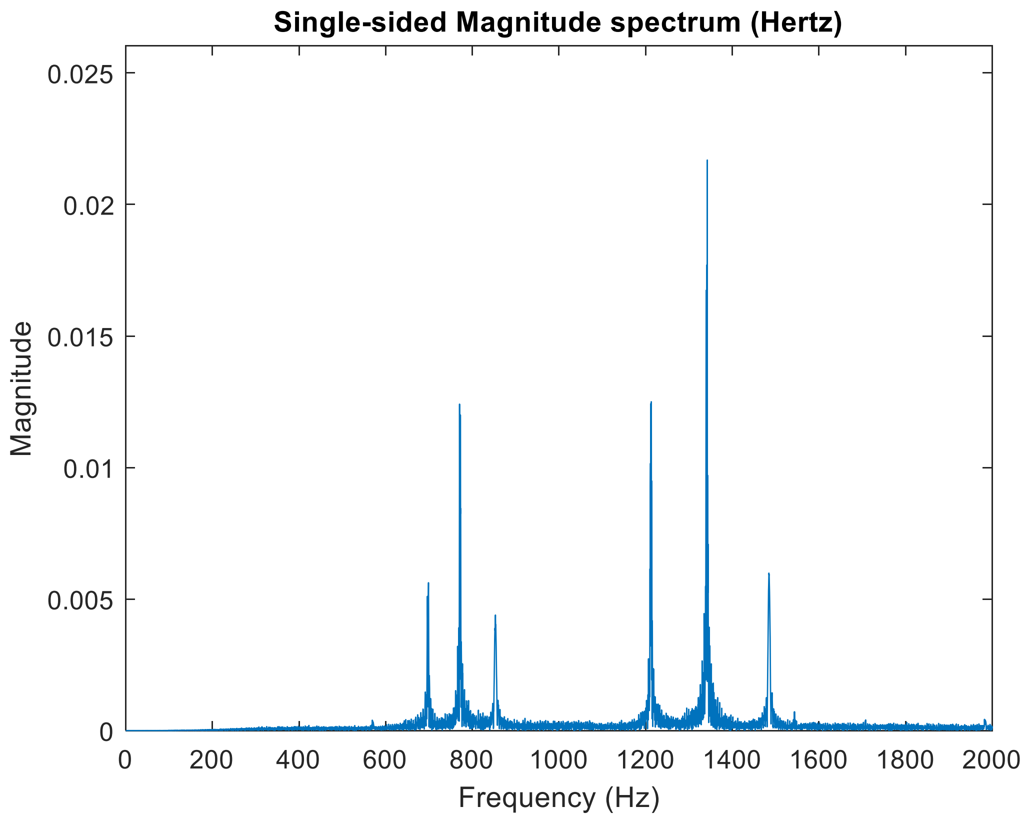
Your task is to read these audio files; in the time-domain separate the individual key presses; and in the frequency-domain identify the number pressed. You should report the phone number dialed for each of these files.

Following is some example code to read in an audio file (`phone_number_4.mp3`) and plot the signal in the time and frequency domains:

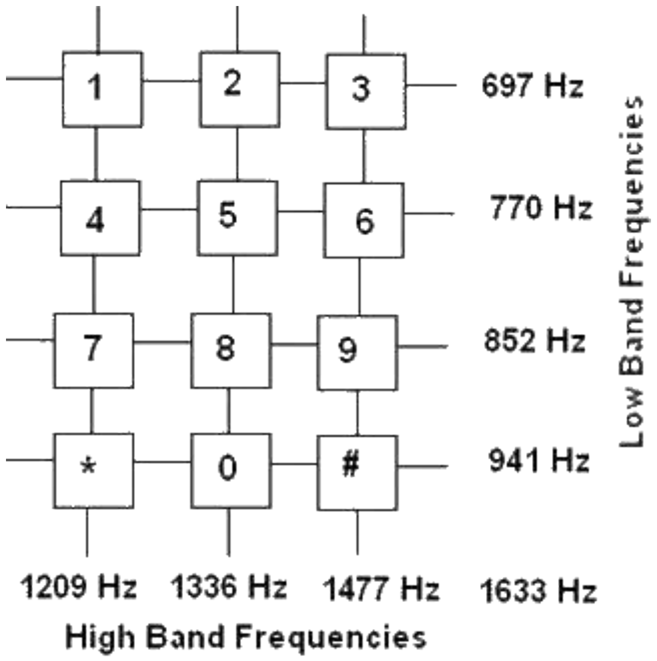
```
phone_number = 'phone_number_4.mp3';
[phn,fs] = audioread(phone_number);
phone_part1 = split(phone_number,'.');
phone_call = split(phone_part1{1},'_');
callnum = phone_call{3};
phn1 = phn(:,1);
nsamp = length(phn1);
tm = (1/fs)*[1:nsamp];
figure(1)
plot(tm,phn1);
xlabel('Time (sec)')
ylabel('Magnitude')
title(['Phone Number ',callnum])

fnyquist = fs/2;
x_mag = abs(fft(phn1))/nsamp;
bins = [0:nsamp-1];
freq_hz = bins*fs/nsamp;
% Plot only positive frequencies
n_2 = ceil(nsamp/2);
figure(2)
plot(freq_hz(1:n_2), x_mag(1:n_2))
max_mag = max(x_mag(1:n_2));
axis([0,2000,0,1.2*max_mag]);
xlabel('Frequency (Hz)')
ylabel('Magnitude');
title('Single-sided Magnitude spectrum (Hertz)');
```

This will create the plots shown on the following page.


Phone Number 4
Magnitude
Time (sec)
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
0
0.5
1
1.5
2
2.5
3
3.5
4


Single-sided Magnitude spectrum (Hertz)
Magnitude
Frequency (Hz)
0.025
0.02
0.015
0.01
0.005
0
0
200
400
600
800
1000
1200
1400
1600
1800
2000

As you can see in the time-domain, there are seven bursts of activity; one for each number pressed. In the frequency-domain, you see a distinct set of frequencies. These are the touchtone frequencies used for Dual-Tone Multi-Frequency signaling (DTMF). Pairs of frequencies map to numbers as shown below:



Your first task will be to partition the time-domain signal into segments corresponding to each key press. Next, you should take the Fourier transform of each segment, identify the pair of frequencies, and determine the corresponding number. I found that Matlab functions like "`movmean`" and "`findpeaks`" were useful for these tasks, but you may find other ways to do this that are better.

For each of the nine numbers, report the frequencies and number pressed in the form:

| # 1 | #2 | #3 | #4 | #5 | #6 | #7 | #8 | #9 |
|---|---|---|---|---|---|---|---|---|
| *f*1 (Hz) | *f*1 (Hz) | *f*1 (Hz) | *f*1 (Hz) | *f*1 (Hz) | *f*1 (Hz) | *f*1 (Hz) | *f*1 (Hz) | *f*1 (Hz) |
| *f*2 (Hz) | *f*2 (Hz) | *f*2 (Hz) | *f*2 (Hz) | *f*2 (Hz) | *f*2 (Hz) | *f*2 (Hz) | *f*2 (Hz) | *f*2 (Hz) |
| Number Pressed | Number Pressed | Number Pressed | Number Pressed | Number Pressed | Number Pressed | Number Pressed | Number Pressed | Number Pressed |

Then, summarize your result as:
Phone number: xxx-xxxx (where the x's are the numbers pressed)

In your report, explain how you accomplished each of these tasks. Include the plots, tables, resulting phone numbers, and code.

**Part B – Word Recognition**

This task involves distinguishing between two words, "yes" and "no." You can imagine this might be part of a phone tree to route a user to a particular banking service, such as "Would you like to hear the balance in your account (Respond yes or no) ?"

This is a classification problem, so we will utilize a tool available in Matlab called "`classificationLearner`."

Compressed directories for training and test data for "yes" and "no" audio files can be found on the class Moodle page at:
Training data for "no" audio files
Test data for "no" audio files
Training data for "yes" audio files
Test data for "yes" audio files

There are 1000 training instances for each word and 200 test instances[2]. (In supervised machine learning it is typical to set aside 20% of the data for blind testing after the classifier has been trained on the training data.) You do not need to use all of these data if you find that the computational resources of your computer are not able to handle this volume of data. For example, you might train on 200 instances and use 40 instances for testing.

Your first task will be to unzip these compressed data into directories for each of these categories. The resulting individual audio files are fairly small (on the order of 30 KB). These are .wav files sampled at 16 kHz. (In the code below, I allow for the sampling rates to be different, just in case, since I could not check every file.)

Example code showing how to read the first 100 files for the training data is shown below:

```
numrec = 100;
D = dir(fullfile('no_training','*.wav'));
for k = 1:numrec
    audiofile = fullfile('no_training',D(k).name);
    [no_array{k},fs_no(k)] = audioread(audiofile);
end
D = dir(fullfile('yes_training','*.wav'));
for k = 1:numrec
    audiofile = fullfile('yes_training',D(k).name);
    [yes_array{k},fs_yes(k)] = audioread(audiofile);
end
```

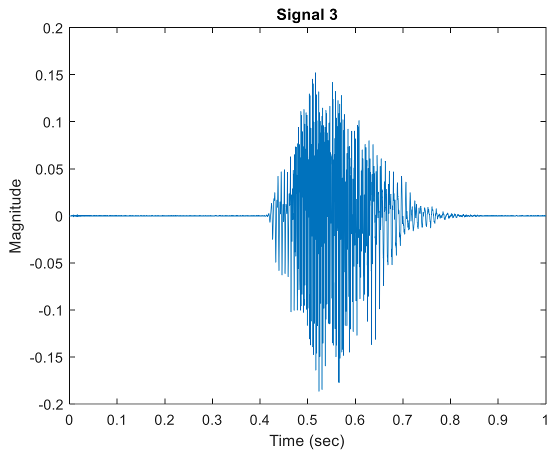
Notice that the files are read into Matlab cell arrays rather than a matrix since the lengths of the recordings may not be the same number of samples. If you want to listen to one of these records, for example no_training record 5, you could use the command:
`sound(no_array{5},fs_no(5))`

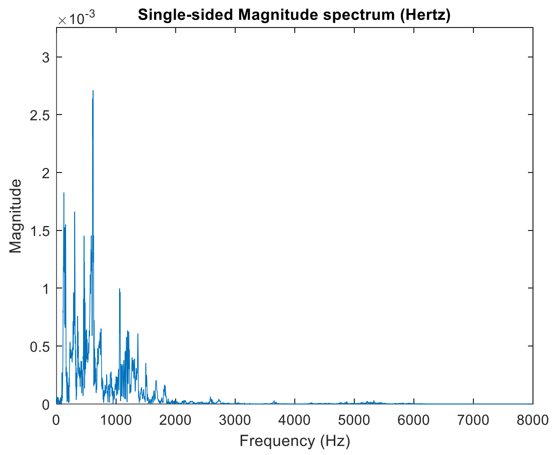
Code to read and plot the time-domain and frequency magnitude spectrum for one no_training instance is shown below:

[2] You may want to combine the training and test data into a single file and let classificationLearner set aside the test data.

```
k = 3;
sig = no_array{k};
fs = fs_no(k);
nsamp = length(sig);
tm = (1/fs)*[1:nsamp];
figure(1)
plot(tm,sig);
xlabel('Time (sec)')
ylabel('Magnitude')
title(['Signal ',num2str(k)])
fnyquist = fs/2;
x_mag = abs(fft(sig))/nsamp;
bins = [0:nsamp-1];
freq_hz = bins*fs/nsamp;
% Plot only positive frequencies
n_2 = ceil(nsamp/2);
figure(2)
plot(freq_hz(1:n_2), x_mag(1:n_2))
max_mag = max(x_mag(1:n_2));
axis([0,2000,0,1.2*max_mag]);
xlabel('Frequency (Hz)')
ylabel('Magnitude');
title('Single-sided Magnitude spectrum (Hertz)');
```

This will create the following two plots:





The major task for this part of the lab is to create a table of features that can be used by the classifier. For example, I divided the spectrum into 10 equal frequency bins, e.g., 0-800 Hz, 801-1600 Hz, etc. I then calculated the energy in each one of these bins and divided by the total energy in the signal (to normalize, since recordings could have quite different gains). I stored these in an array that is in the Matlab workspace at the end of processing. (It could also be written it out as a table or spreadsheet to be read in by another program.) The structure of the array looks as follows for the first four instances:

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| E1,1 | E1,2 | E1,3 | E1,4 | E1,5 | E1,6 | E1,7 | E1,8 | E1,9 | E1,10 | 0 or 1 |
| E2,1 | E2,2 | E2,3 | E2,4 | E2,5 | E2,6 | E2,7 | E2,8 | E2,9 | E2,10 | 0 or 1 |
| E3,1 | E3,2 | E3,3 | E3,4 | E3,5 | E3,6 | E3,7 | E3,8 | E3,9 | E3.10 | 0 or 1 |
| E4,1 | E4,2 | E4,3 | E4,4 | E4,5 | E4,6 | E4,7 | E4,8 | E4,9 | E4,10 | 0 or 1 |

There is a row for each instance (recording) and a column for each attribute, in this case, the energy in frequency bins 1 through 10. Additionally, the last column is the class of the instance (0 for no, 1 for yes).

If you construct a table of features in this way, it can be used in the classificationLearner, since it has access to whatever data is in the Matlab workspace.

To start the classificationLearner tool, at the Matlab prompt type:

```
>> classificationLearner
```

I will demonstrate how to use this tool in class.

ClassificationLearner can set up cross-validation and testing. (I will explain these in class.)

You should explore different methods of creating features to be used by the classifier.

The final result for this part of the lab will be the classification accuracies on the training data (with 5-fold cross-validation) and test data.

Report on your method of selecting features and the classification accuracy obtained.