EE679: Computing Assignment 4

Swapnil Bembde 14D070034

November 4, 2017

Automatic Segmentation

This is the necessary step for further parts. To extract an uttered digit a method based on short term energy is employed. Some thresholding is also done to discriminate the digit utterance. To reduce the glottal roll-off pre-emphasis has been done. Resulted segmented sounds are stored in '/segmented/' folder.

Listing 1: Code Snippet: Segmentation

```
%pre-emphasis
data_inp = data_inp - 0.95 * [0; data_inp(1:size(data_inp,1)-1)];
%energy calculation
energy = data_inp .^ 2;
summ = ones(1, winlen);
energy = conv(energy, summ);

ptr = 1;
for k=0:9
    k;
    while energy(ptr) < threshold
        ptr = ptr + 1;
    end
    first = ptr;
    while energy(ptr) >= threshold
        ptr = ptr + 1;
    end
    last = ptr;
    digit_data = data_inp(max(1, first - overlap1):min(last-overlap2, size(data_inp,
        1)));
    audiowrite(strcat(filename(21:end-4), num2str(k), '0.wav'), digit_data, fSampling
        );

    while energy(ptr) < threshold
        ptr = ptr + 1;
    end
    first = ptr;
    while energy(ptr) >= threshold
        ptr = ptr + 1;
    end
    last = ptr;
    digit_data = data_inp(max(1, first - overlap1):min(last-overlap2, size(data_inp,
        1)));
    audiowrite(strcat(filename(21:end-4), num2str(k), '1.wav'), digit_data, fSampling
        );

end
```

MFCC Calculation

Steps:
*Divide the digit into short frames.
*Calculate the power spectrum of each frame.
*Multiply by mel filterbank (40 is the maximum number of filters).
*Take logarithm of the energies calculated.
*Then take DCTs and keep 1-13 coefficients.
These 13 coefficients represent a frame for identification.

Listing 2: Code Snippet: MFCC calculator

```
function mfcc = calcMFCC(input_data, fSampling, lowf, highf, nFilts, fftSize)

    FrameSize = fSampling/100; %framesize = 10 ms
    nVectors = floor(size(input_data, 1)/FrameSize) - 1;
    mfcc = zeros(13, nVectors);

    melFiltBank = melFilter(nFilts, lowf, highf, fSampling, fftSize);

    for i = 1:nVectors
        %windowed
        win = hamming(FrameSize).*input_data(((i-1)*FrameSize/2+1):((i-1)*FrameSize
            /2+FrameSize));

        %energy calculation
        winFFT = abs(fft(win, fftSize));
        winFFT = winFFT(1:size(melFiltBank, 2));
        FrameEnergy = (winFFT.^2)/(fftSize);

        melEnergy = log(melFiltBank*FrameEnergy);

        allCoeffs = dct(melEnergy);
        mfcc(:, i) = allCoeffs(1:13);

    end
end
```

Digit Recognizer

Testing using training vectors directly

First of all for training, in leave-one-speaker-out mode 640 words were analyzed. For each frame we assign MFCC vector and MFCC index. MFCC index has a digit corresponding to the MFCC vector.

For testing, we run k-nearest neighbors algorithm. This finds closest match to our input MFCC vector. Different distance criterion were tried and found that euclidean distance gives minimum WER for this application.

With this method, we get a WER = 17.8%. The common confusions are represented by this table -

Table 1: Confustion Matrix

I & O	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	53	0	0	9	0	0	0	0	2	0
1	0	61	0	0	1	2	0	0	0	0
2	2	3	54	0	5	0	0	0	0	0
3	11	0	0	44	0	1	0	0	8	0
4	0	4	4	0	56	0	0	0	0	0
5	0	9	0	0	1	50	0	2	0	2
6	3	0	0	0	2	6	39	8	6	0
7	0	0	0	0	0	6	0	58	0	0
8	1	0	0	0	0	4	6	0	53	0
9	1	1	0	1	0	2	0	1	0	58

Sum of all the entries = 640 and Sum of the all entries in a row = 64 (16-fold cross-validation)
Rows represent the input digits and columns represent the predicted digits. An entry C(i,j) means input digit is i-1 and it is predicted as j-1 this many times during testing.
From the above matrix, there are some common confusions between zero and three. Sometimes, five is recognized as one and three as eight.
According to our recognizer, the most difficult digit to recognize is six.

Listing 3: Code Snippet: Digit Recognizer based on the “bag of frames”

```
%clc;
%workspace;
%close all;
speakers = { 'Mayur','Shrikant','Vedhas', 'Niramay',...
    'divyansh','Jatin', 'Hitesh', 'Kaustuv', 'Ankita',...
    'Kamini','Mansi','Pragya','Prarthana','Reshma',...
    'Richa','Shradha'};
lowf = 150;
highf = 7000;
nFilt = 40;
fftSize = 1024;

Tested = 0;
incorrect = 0;
%rows predicted(output) class & columns target class
confusionMatrix = zeros(10, 10);

for leftOut = 1:16

    MfccVectors = [];
    MfccIndex = [];

    for speakerIndex = 1:16
        if speakerIndex ~= leftOut
            for recordings = 1:2
                for digit = 0:9
                    for utterance = 0:1
                         % Training
                        [extracted_data, fSamping] = audioread(strcat('bof/',
                            speakers{speakerIndex}, num2str(recordings), num2str(digit
                            ), num2str(utterance), '.wav'));
                        mfccs = calcMFCC(extracted_data, fSamping, lowf, highf, nFilt
                            , fftSize);
                        MfccVectors = [MfccVectors; mfccs'];
                        MfccIndex = [MfccIndex; repmat(digit, size(mfccs, 2), 1)];

                    end
                end
            end
        end
    end
    disp('Trained!!!');
    for recordings = 1:2
        for digit = 0:9
            for utterance = 0:1
                % Testing
                [extracted_data, fSampling] = audioread(strcat('bof/', speakers{
                    leftOut}, num2str(recordings), num2str(digit), num2str(utterance),
                     '.wav'));
                mfccs = calcMFCC(extracted_data, fSampling, lowf, highf, nFilt,
                    fftSize);
                mfccRec = knnsearch(MfccVectors, mfccs');
                PredictedDigit = mode(MfccIndex(mfccRec));

                % Results
                Tested = Tested + 1;
                incorrect = incorrect + (PredictedDigit ~= digit);

                % confusion matrix
                confusionMatrix(1+digit, 1+PredictedDigit) = confusionMatrix(1+digit,
                    1+PredictedDigit) + 1;

            end
        end
    end

end

wer = sum(incorrect)/sum(Tested) * 100; disp(wer);
```

Vector Quantization based on Clustering

First of all for training, in leave-one-speaker-out mode 640 words were analyzed. For each frame we assign MFCCvector and MFCC index. MFCC index has a digit corresponding to the MFCC vector.
Instead of entire training vectors, we find K-clusters of MFCC vectors using kmeans for each digit. Number of clusters is chosen depending on the accuracy result. For this system I have chosen k = 80 due to convergence problem and accuracy improvement.
For testing, we run k-nearest neighbors algorithm. This finds closest match to our input MFCC clustering centroid. Each digit is represented by 80 centroids of length 13.
By this approach, resulting WER is 20.93%. The common confusion is stated by the matrix below.

Table 2: Confusion Matrix

I & O	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	48	0	0	9	0	0	4	0	3	0
1	0	61	0	0	1	2	0	0	0	0
2	0	2	59	0	1	0	0	1	1	0
3	7	0	0	44	0	0	4	0	9	0
4	1	4	3	0	56	0	0	0	0	0
5	0	12	0	0	4	39	0	3	1	5
6	4	0	0	2	2	3	35	11	7	0
7	0	0	0	0	0	2	1	59	0	2
8	2	0	0	0	0	1	7	0	53	1
9	0	2	0	4	0	2	0	4	0	52

Sum of all the entries = 640 and Sum of the all entries in a row = 64 (16-fold cross-validation)
Rows represent the input digits and columns represent the predicted digits. An entry C(i,j) means input digit is i-1 and it is predicted as j-1 this many times during testing.
From the above matrix, some common confusions are 1. zero and three difficult to recognize. 2. Sometimes three is recognized as eight. 3. five is recognized as one. 4. A few times six is recognized as seven. Six is the most difficult digit to recognize.

It is expected that if we increase the number of clusters, WER will decrease. But this will increase our computational complexity. We can't expect very low WER in BOF and VQ because our recognition is not phoneme based. If words have the same phonemes, then they are difficult to discriminate.

Listing 4: Code Snippet: Digit Recognizer based on Vector Quantization

```
clc;
workspace;
close all;
speakers = { 'Mayur','Shrikant','Vedhas', 'Niramay',...
    'divyansh','Jatin', 'Hitesh', 'Kaustuv', 'Ankita',...
    'Kamini','Mansi','Pragya','Prarthana','Reshma',...
    'Richa','Shradha'};
lowf = 150;
highf = 7000;
nFilt = 40;
fftSize = 1024;
nClusters = 80;

Tested = 0;
incorrect = 0;
%rows predicted(output) class & columns target class
confusionMatrix = zeros(10, 10);

for leftOut = 1:16

    MfccVectors = [];
    MfccIndex = [];

    for digit = 0:9
        for recordings = 1:2
            for speakerIndex = 1:16
                if speakerIndex ~= leftOut
                    for utterance = 0:1

                         % Training
                        [extracted_data, fSampling] = audioread(strcat('bof/',
                           speakers{speakerIndex}, num2str(recordings), num2str(digit
                           ), num2str(utterance), '.wav'));
                        mfccs = calcMFCC(extracted_data, fSampling, lowf, highf,
                           nFilt, fftSize);
                        MfccVectors = [MfccVectors; mfccs'];
                        MfccIndex = [MfccIndex; repmat(digit, size(mfccs, 2), 1)];

                    end
                end
            end
        end
    end
    disp('Trained!!!')
    % Vector Quantization using kmeans
    MfccVectorsK = zeros(10*nClusters, 13);
    MfccIndexK = zeros(10*nClusters, 1);

    for digit = 0:9
        tempIndex = find(MfccIndex == digit);
        tempVectors = MfccVectors(tempIndex(1):tempIndex(end), :);

        [~, centers] = kmeans(tempVectors, nClusters);%,'MaxIter',1000);
```

```matlab
        MfccVectorsK((digit*nClusters+1:digit*nClusters+nClusters), :) = centers;
        disp([strcat('vectors ',num2str(digit),num2str(leftOut))]);
        MfccIndexK(digit*nClusters+1:digit*nClusters+nClusters) = digit;

    end
    MfccVectors = MfccVectorsK;
    MfccIndex = MfccIndexK;
    disp('Quantized!!!')

    for recordings = 1:2
        for digit = 0:9
            for utterance = 0:1
                % Testing
                [extracted_data, fSampling] = audioread(strcat('bof/', speakers{leftOut}, num2str(recordings), num2str(digit), num2str(utterance), '.wav'));
                mfccs = calcMFCC(extracted_data, fSampling, lowf, highf, nFilt, fftSize);
                mfccRec = knnsearch(MfccVectors, mfccs');
                PredictedDigit = mode(MfccIndex(mfccRec));

                % Results
                Tested = Tested + 1;
                incorrect = incorrect + (PredictedDigit ~= digit);

                % confusion matrix
                confusionMatrix(1+digit, 1+PredictedDigit) = confusionMatrix(1+digit, 1+PredictedDigit) + 1;

            end
        end
    end

end

wer = sum(incorrect)/sum(Tested) * 100; disp(wer);
```

Recognition using Dynamic Time Warping

This is a non-probabilistic method with benefit of dynamic programming to match a template against incoming signal. This stores the utterances in a specific manner to improve accuracy of the recognizer. By training our system learns feature vectors corresponding to each frame and then these feature vectors are matched to the input feature vector using DTW distance criteria with respect to MFCCs.
For this approach a WER is 6.87%. The confusion matrix is shown below.

Table 3: Confusion Matrix

I & O	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	52	0	0	3	0	0	2	0	5	2
1	0	63	0	0	0	1	0	0	0	0
2	0	0	62	1	0	0	0	0	1	0
3	2	0	0	60	0	0	1	0	1	0
4	0	0	1	0	63	0	0	0	0	0
5	0	3	0	0	0	58	0	1	0	2
6	0	0	0	0	0	0	64	0	0	0
7	0	0	0	0	0	2	9	50	2	1
8	0	0	0	1	0	0	0	2	61	0
9	0	0	0	1	0	0	0	0	0	63

Sum of all the entries = 640 and Sum of the all entries in a row = 64 (16-fold cross-validation)
Rows represent the input digits and columns represent the predicted digits. An entry C(i,j) means input digit is i-1 and it is predicted as j-1 this many times during testing.
Some of the common confustions are: 1. confusion between zero and three. 2. Sometimes zero is recognized as eight and seven as six. 3. Most difficult digit to recognize using our system is seven.

Using this approach, we get much better confusion matrix and much lower WER as compared to direct training and VQ based approaches. This is due to this approach is based on capturing the time of each phoneme. Here, there is a specific sequence of the time information of phonemes. Hence, the confusion between digits is less than earlier methods.

Listing 5: Code Snippet: Digit Recognizer based on template matching

```
clc;
workspace;
close all;
speakers = { 'Mayur','Shrikant','Vedhas', 'Niramay',...
    'divyansh','Jatin', 'Hitesh', 'Kaustuv', 'Ankita',...
    'Kamini','Mansi','Pragya','Prarthana','Reshma',...
    'Richa','Shradha'};
lowf = 150;
highf = 7000;
nFilt = 40;
fftSize = 1024;

Tested = 0;
incorrect = 0;
%rows predicted(output) class & columns target class
confusionMatrix = zeros(10, 10);

for leftOut = 1:16

    complete_data = {};
    ids = [];
    ptr = 1;

    for speakerIndex = 1:16
        if speakerIndex ~= leftOut
            for recordings = 1:2
                for digit = 0:9
                    for utterance = 0:1

                        %Training
                        [extracted_data, fSampling] = audioread(strcat('bof/',
                            speakers{speakerIndex}, num2str(recordings), num2str(digit
                            ), num2str(utterance), '.wav'));
                        disp(strcat('Training/', speakers{speakerIndex}, num2str(
                            recordings), num2str(digit), num2str(utterance), '.wav'));
                        complete_data{ptr} = extracted_data;
                        ids(ptr) = digit;
                        ptr = ptr + 1;

                    end
                end
            end
        end
    end


    disp('Trained!!!');

    % Testing
    for recordings = 1:2
        for digit = 0:9
            for utterance = 0:1

                [extracted_data, fSampling] = audioread(strcat('bof/', speakers{
                    leftOut}, num2str(recordings), num2str(digit), num2str(utterance),
                    '.wav'));
                disp(strcat('Testing/', speakers{leftOut}, num2str(recordings),
                    num2str(digit), num2str(utterance), '.wav'));

                minDistIds = 0;
                minDistance = Inf;
                for i = 1:size(complete_data, 2)

                    disp([strcat('checking/',speakers{leftOut}, num2str(recordings),
                        num2str(digit), num2str(utterance), '.wav')]);

                    tempDistance = DTWdist(extracted_data, complete_data{i},
                        fSampling, lowf, highf, nFilt, fftSize);
                    if tempDistance < minDistance

                        minDistance = thisDist;
                        minDistIds = i;

                    end

                end
                PredictedDigit = ids(minDistIds);

                % Results
                Tested = Tested + 1;
                incorrect = incorrect + (PredictedDigit ~= digit);

                % confusion matrix
                confusionMatrix(1+digit, 1+PredictedDigit) = confusionMatrix(1+digit, 1+PredictedDigit) + 1;

            end
        end
    end
end
wer = sum(incorrect)/sum(Tested) * 100; disp(wer);
```

Listing 6: Code Snippet: Distance between signals using DTW

```
function distanceSum = DTWdist(inp1, inp2, fSampling, lowf, highf, nFilts, fftSize)

    mfccinp1 = calcMFCC(inp1, fSampling, lowf, highf, nFilts, fftSize);
    mfccinp2 = calcMFCC(inp2, fSampling, lowf, highf, nFilts, fftSize);

    %This is the function directly gives the output
    %Rather than the script this function works faster
    %%[distanceSum,~,~] = dtw(mfccinp1,mfccinp2);
    n = size(mfccinp1, 2);
    m = size(mfccinp2, 2);
    distanceMatrix = zeros(n, m);
    for i = 1:n
        for j = 1:m

            distanceMatrix(i, j) = norm(mfccinp1(:, i) - mfccinp2(:, j));
        end
    end

    DyanmicWarping = zeros(n+1, m+1);

    for i = 2:n+1
        DyanmicWarping(i, 1) = Inf;
    end
    for i = 2:m+1
        DyanmicWarping(1, i) = Inf;
    end
    DyanmicWarping(1, 1) = 0;

    for i = 2:n+1
        for j = 2:m+1

            cost = distanceMatrix(i-1, j-1);
            DyanmicWarping(i, j) = cost + min(DyanmicWarping(i-1, j), min(
               DyanmicWarping(i-1, j-1), DyanmicWarping(i, j-1)));

        end
    end

    distanceSum = DyanmicWarping(n+1, m+1);

end
```