Homework 5: Music Classification with SVD

Karl Marrett (kdmarrett@gmail.com)

Abstract

In this report, we use the Singular Value Decomposition (SVD) in order to finger print the spectogram of different songs, bands, and genres. By taking the spectogram of songs, which gives the changes in frequency over time, we keep data that accentuates changes in the frequency range, tempo, beat, etc. of our music. Furthermore, by using SVD we elucidate another method for taking high dimensional data and projecting it down into a smaller rank and perhaps more meaningful basis. With this approach, we test the ability of a clustering technique known as Linear Discriminant Analysis (LDA) to classify the song data. In this report the goals are threefold:

1. Use the SVD to train the classification of the different bands Neon Indian, Vitalic, and Arthur Grumiaux and test the accuracy in classifying new samples from these bands.
2. Use the SVD to classify and test the accuracy of discriminating three early 90's alt-rock bands Sonic Youth, The Pixies, and Yo La Tengo.
3. Use the SVD to classiy and test the accuracy of discriminating the three music genres of R'n'B, Shoegaze, and Gregorian Chants.

1 Introduction

Finding the right basis and the methods to transform data into such a basis are central problems in machine learning. In the past reports, we have mostly used certain assumptions about our data to decide a reasonable basis to transform our data. For example, in the analysis of sound files we used the convenient decomposition into the Fourier domain to create a low dimensional picture of the dynamics of the data.[2] However, when we instead don't know anything about the underlying structure of mechanism of the data, we need a systematic approach. For example, given neural data of a spike train response of several neurons we may be able to explain structure in the data in a low dimensional Fourier space but the simplest model may in fact be described in an entirely different space. For problems of this nature, we want a data-driven approach for choosing the right basis to describe our data. Having a generic to high dimensional data of this sort allows equation-free modeling while also sidestepping our potentially misleading initial assumptions about the data.

These concerns illustrate some of the motivation for new methods for choosing a lower dimensional space for the data. One approach of choosing a basis involves the technique known as Singular Value Decomposition, denoted as SVD henceforth in the report. The information that the SVD produces is quite similar to the eigen decomposition of the covariance matrix used in the previous report, in that the principal modes of the matrix are returned. Further details of how this technique might describe a low dimensional bases for our data is described in detail in the next section.

2 Theoretical Background

2.1 SVD

The main means of compressing and characterizing the high dimensional song data is via the SVD technique. The theorem introducing the SVD essentially asserts that *any* matrix can be decomposed as follows:

$$A_m n = U_{mm} S_{mn} V_{nn}^T$$

The decomposition has deep connections to the eigen faces of A, namely that the columns of U are orthonormal eigenvectors of AA^T and that the columns of V are also the orthonormal eigenvectors of $A^T A$.[1] Another point to note is that the diagonal matrix S holds the square roots of the eigenvalues of U or V. These relations allow us to analytically solve for the U, S, or V matrices although in this report we will only be leveraging the Matlab command *svd*. Since U represents the new modes of A we can treat SV^T as the projection on to the modes. We take these projections as the means to classify the song data. When implementing the SVD in the script we need to use the reduced SVD to the size of the data in order to have reasonable processing time. This means that if $m > n$ in the A matrix, in the reduced form of the svd, only the first n columns are computed, thus S is n-by-n in dimension.

The major advantage of the SVD is the ordered components of the weights of the S matrix, which gives a criterion for deciding which dimensions describe meaningful structure in the data, and those which are redundant. This is used as a further method for distilling the meaningful information from the song data.

2.2 LDA

Another major technique used in the report was Linear Discriminant Analysis (LDA). LDA is a technique originally developed by Fischer, an eminent statistician.[2] It has become a classic technique in pattern recognition for its ability to project data into a new bases that maximally discriminates them. It essentially chooses a basis that maximizes the variance between the means of the two categories and clusters the data accordingly. Expressed formally in equation 1 below:

$$\mathbf{w} = \text{argmax} \frac{\mathbf{w^T S_B w}}{\mathbf{w^T S_W w}} \tag{1}$$

where the two quantities $\mathbf{S_B}$ and $\mathbf{S_W}$ are defined as:

$$\mathbf{S_B} = (\mu_\mathbf{1} - \mu_\mathbf{2})(\mu_\mathbf{2} - \mu_\mathbf{1})^\mathbf{T} \tag{2}$$

$$\mathbf{S_B} = \sum_{j=1}^{2} \sum_{\mathbf{x}} (\mathbf{x} - \mu_\mathbf{j})(\mathbf{x} - \mu_\mathbf{j})^\mathbf{T} \tag{3}$$

This analysis was performed in the report using the Matlab command *classify*.

3 Algorithm Implementation and Development

As alluded in the theoretical background section, most of the mathematical sophistication of the identification techniques was handled by the *svd*, *classify*, and also the *spectogram* functions in Matlab. This means that the brunt of the complexity of the code was dealing with cleaning and ordering the data accordingly. For each of the three sections of the report, the music files were put into a characteristic folder. The function *getSpec* written for this project conveniently batch processed every music file in the directory given. Two separate bash scripts were written to convert music files from my local library to wav files and to keep all of the sampling frequencies for every song the same. Processing the sound files in a separate function also allowed clearance of variables so that memory loads did not occur. For every sound file in the directory, a random sampling of three second clips was taken. From these files, the function *spectogram* was used to compute the changes in frequency over time.

My gabor function developed from earlier reports was originally used but the timing of the matlab embedded *spectogram* was better, so it was used in the final analysis. Using spectogram data accentuates certain characteristics of the music. For example, while the raw trace would capture timing and patterns in beat, it would not capture spectral information such as the pitch range of the band or genres as strongly, which is likely key for a discrimination task of this sort. For these reasons, a spectogram was an ideal preprocessing step to give the SVD decent traction on the songs provided. During this process the song was downsampled to save space and the data was reshaped so that each clip became essentially one column to deal with. Treating a clip as a single vector made computations further in the script much easier. For each of the three categories of music, a variable number of usable vectors were produced but in order to prevent over including of one category in the classification task, a constant number from each group was needed. To combat this, I took the minimum number of samples from the three groups for the rest of the computations. The data returned by *getSpec* was a matrix with the rows defined by the spectogram of a song sample with a single column for each clip.

Each of the sets of columns for each part was concatenated then run using the reduced SVD which returned the modes u, the singular values s and the data's projections onto the modes v. For the classification remainder of the script, the corresponding elements of the projection vectors (named v in the script) were split into three respective categories. Of these

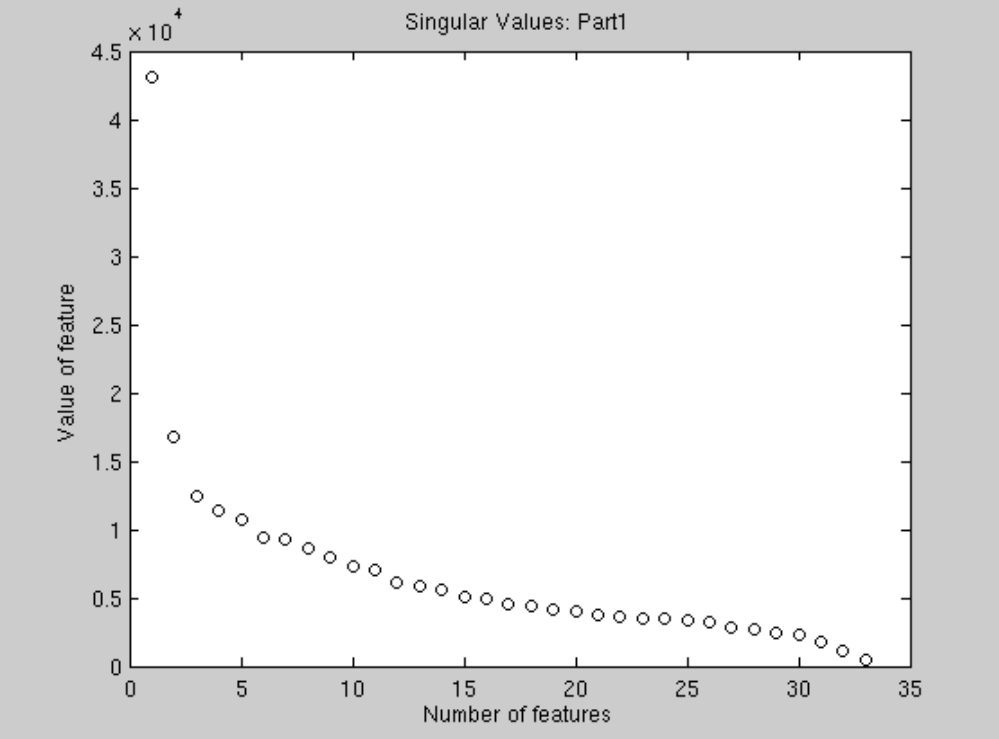

Figure 1: This figure shows the singular values for the first part of the experiment using a smaller data set of an album each from each band type. The dominant mode indicates the preprocessing steps have been done ideally to help the classification accuracy.

categories, the trials were split in training and test trials which we were then classified using the *classify* Matlab command. To compute accuracy of the classification task I compared the values returned by the *classify* function with ideal results. By computing a total error for each trial and running multiple times within a for loop to average these errors, I was able to cross validate my approach.

As a side note, included in the main script *SVDScript.m* was also numerous points of diagnostics. However, since printing to the screen is expensive these were designed with the ability to be suppressed.

To verify changes in accuracy due to method vs. chance, three different approaches were taken for each of the parts. For the first part, a smaller amount of training samples was used to see the robustness of the algorithm given a small N. In the second part three very similar sounding bands were chosen to make the task more difficult. In the final case a large amount of data was used and the genres were as distinct as possible to test the upper limits on the accuracy of classification.

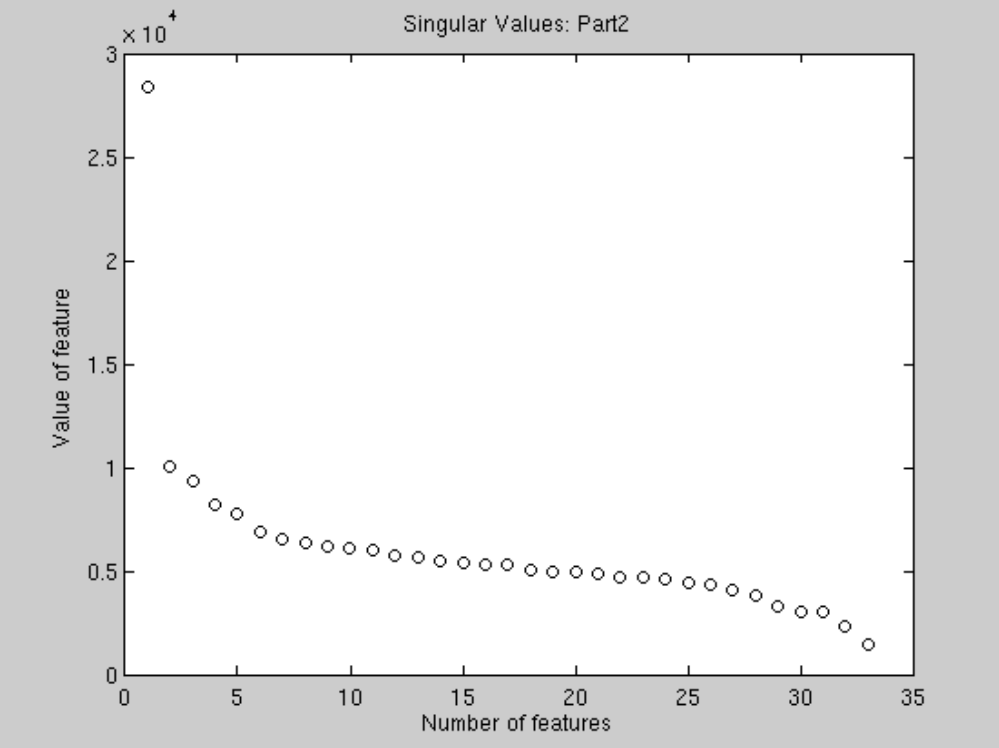

Figure 2: This figure shows the singular values for the second part of the experiment comparing the songs of three bands from one genre. In this case the genre was early 90s alt-rock with the bands Yo La Tengo, Sonic Youth, and The Pixies.

4 Computational Results

The accuracy results are shown in Figure 1. We have some surprising results for the second part using bands of the same genres. It turns out that the accuracy of classification in this case was consistently higher than the three different bands in this first part. This can be attributed to the higher number of total songs and thus clips used in this case. The final part that used many songs boasted the highest accuracy with an average of about .63. From these results we can see that the biggest factor effecting the classification accuracy is actually the number clips used rather than the underlying genre or band used. This means that perhaps with more processing power a higher accuracy for all three conditions could be achieved.

We can also see the singular values of the data matrix in the three different conditions. A strong first component is visible in all parts of the experiment. These strong primary modes which explains why we have reasonable classification accuracy across all parts of the experiment.

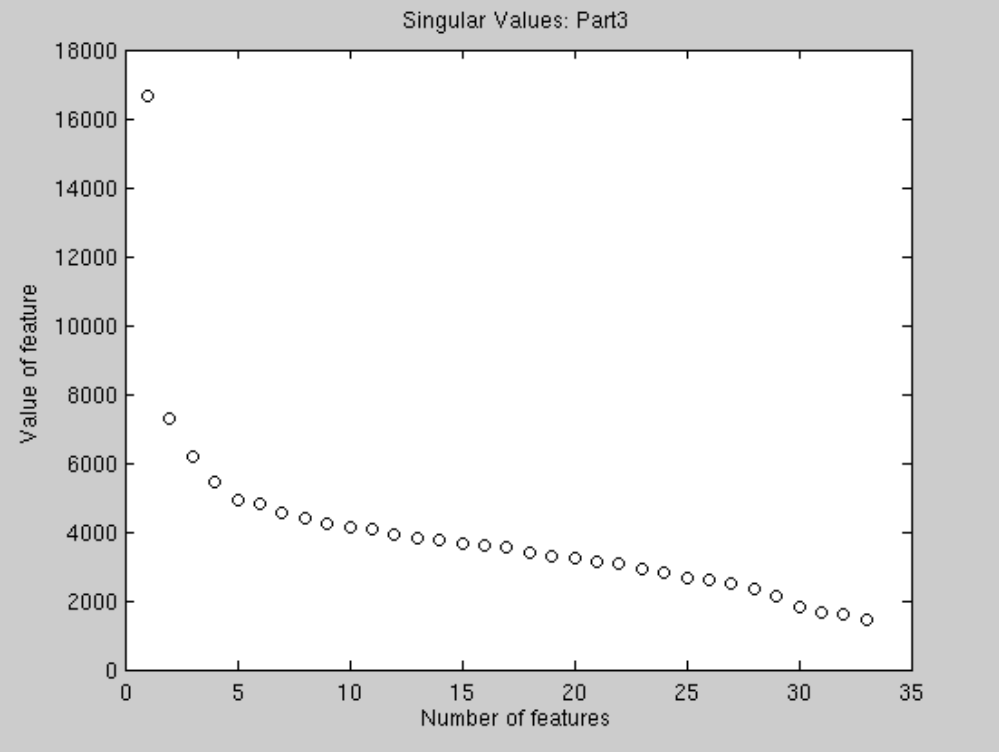

Figure 3: This figure shows the singular values for the third part of the experiment comparing many different songs of three genres. In this case, the genre was chosen to have maximal difference from the others so Gregorian chanting, classical violin, and RnB songs were used.

Table 1: Table showing the average error across different clustering techniques used for the three different conditions.

	Case 1	**Case 2**	**Case 3**
Error by Projection	.39	.47	.63

5 Conclusion

These techniques show the core power of using SVD as a start to a basic machine learning problem. This approach has the advantage that it can identify an ordered low rank approximation of a matrix without relying on assumptions of the underlying behavior of the system besides it being nonlinear.

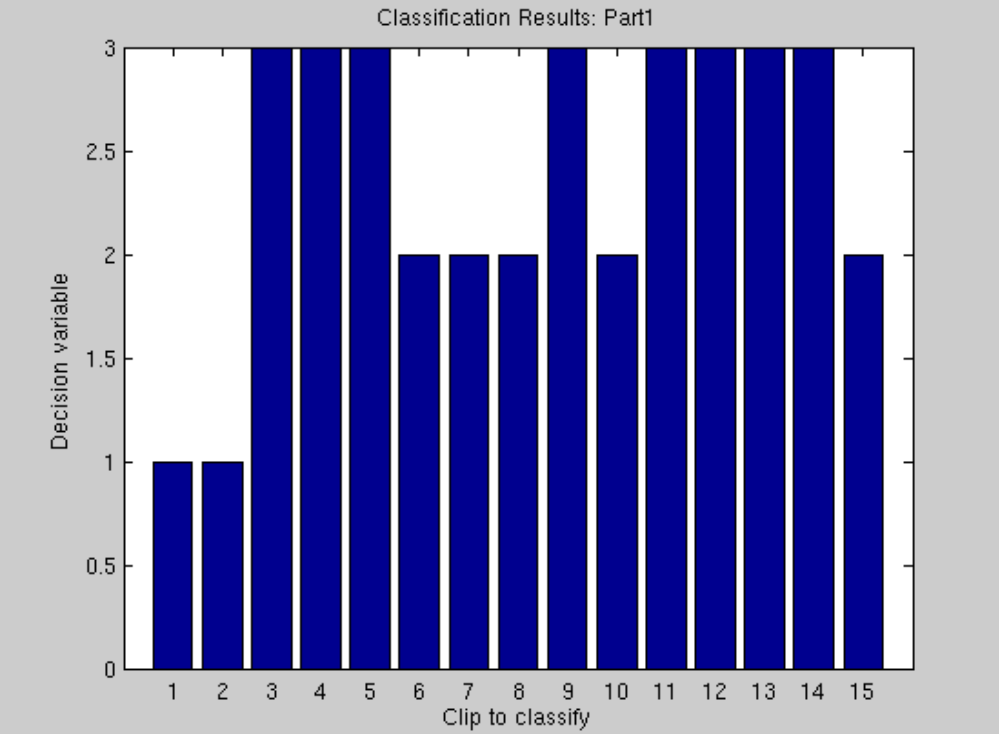

Figure 4: A histogram displaying the decision of the classify algorithm. The value (y axis) shows the decision whereas the x axis shows the different conditions. For this set of trials the correct classification would be 1 for the left third clips 2 for the middle and three for the right third of the plot. Any deviations from these values were used to cross validate the error of the classification method.

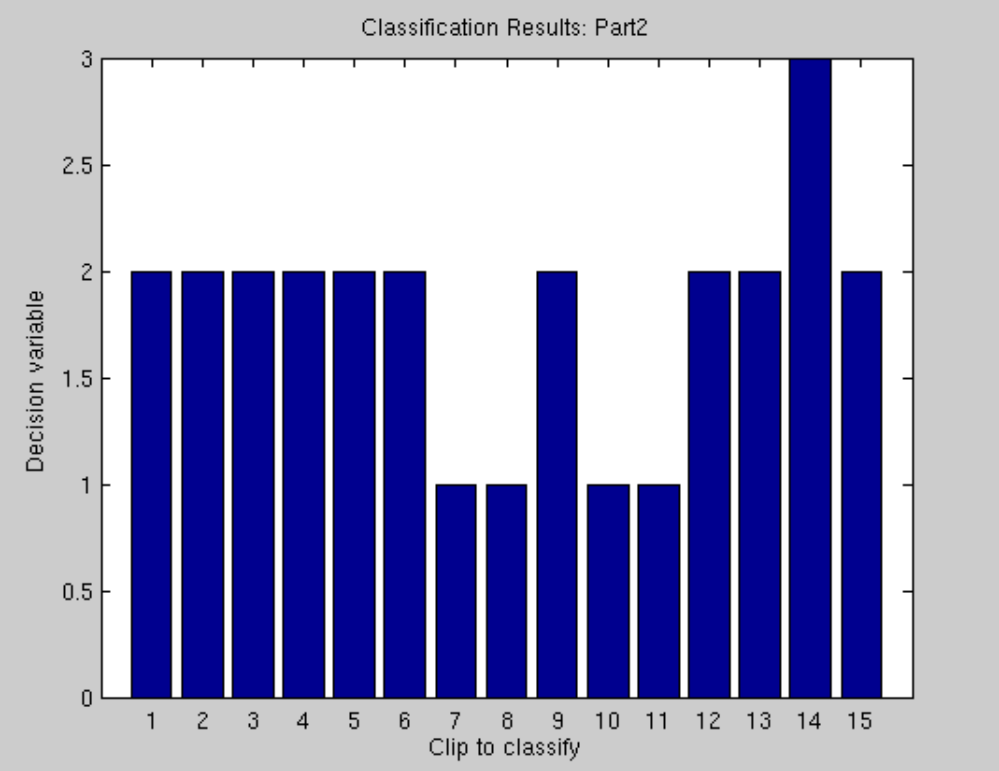

Figure 5: A histogram displaying the decision of the classify algorithm. The value (y axis) shows the decision whereas the x axis shows the different conditions. In this plot the left third of clips were correctly identified as 2, the middle were ideally to be classified as 1 and the right third were meant to be correctly classified as 3.

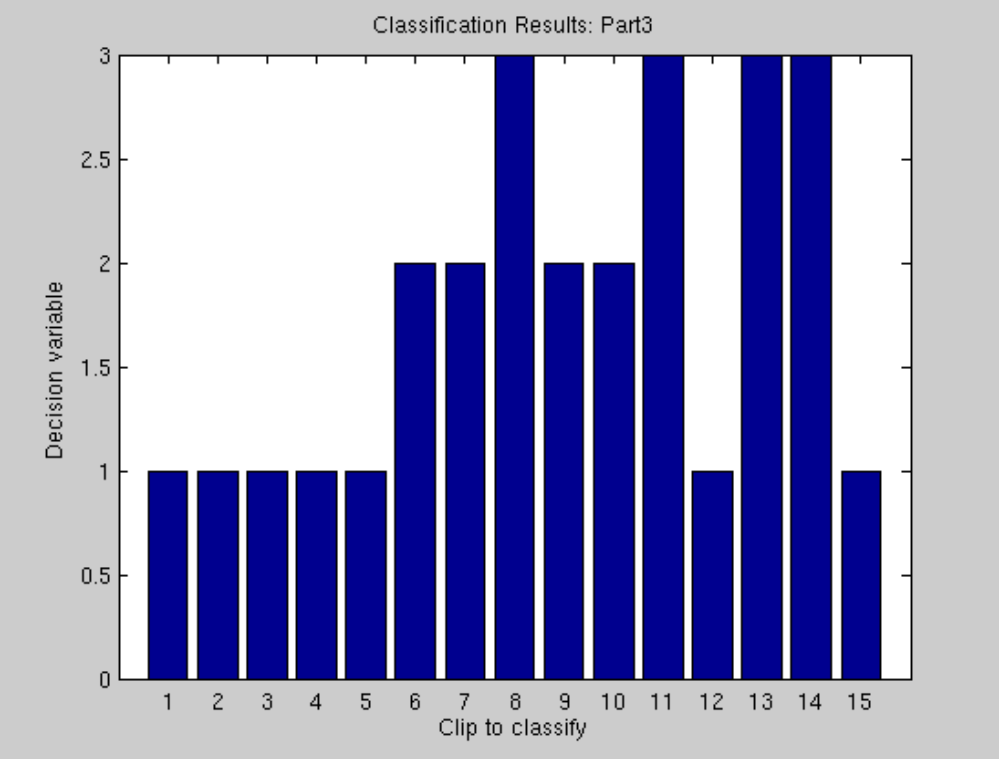

Figure 6: A histogram displaying the decision of the classify algorithm. The value (y axis) shows the decision whereas the x axis shows the different conditions. In this round of trials the first third were meant to be classified as 1, the middle as 2, and the right third as 3. As shown, the classification accuracy is fairly high.

A

[u,s,v] = svd(A, OPTION) Computes the singular value decomposition for the given matrix. In this report we use the option flag '0' to specify the reduced SVD.
classify Matlab function for conducting linear discriminant analysis on the rows of three passed arguments.
spectogram Matlab function for the short-time Fourier transform used in this analysis for returning the one-sided Fourier transform across several points across a clip of the song

B

```
%Karl Marrett
%AMATH 482
%Homework 5 SVD
%Due March 12th 2015
%SVDScript.m

close all; clear all;

trials = 50;
clipNum = 1;
trainingFraction = .50;
features = 10;

createFigures = 1;
% printing bools
songProcessing = 0;
printProcessing = 1;

% directories of music for each part
parts{1} = {'violin', 'neonindian', 'vitalic'};
parts{2} = {'sonicyouth', 'yolatengo', 'pixies'};
parts{3} = {'greg', 'shoegaze', 'RnB'};

mincols = 9999; % dummy value
for part = 1:length(parts)
    for i = 1:trials
        % three directories with which to classify contained wav files
        rng('shuffle');
        threeDirs = parts{part};
        [dat, mincols] = getSpec(threeDirs, clipNum, songProcessing, ...
            mincols);
```

```
    % Arrange data
    D = [];
    for k = 1:length(threeDirs)
        % concatenate categories horizontally
        %mincols
        D = [D, dat.(threeDirs{k})(:, 1:mincols)];
    end

    % square V to be small direction not huge direction
    [u,s,v] = svd(D, 0); % reduced svd

    trainTrials = round(trainingFraction * mincols);
    testTrials = mincols - trainTrials;
    [accuracyP(i), classP ] =...
        classifySVD(u, s, v, trainTrials, ...
        testTrials, mincols, threeDirs, features);
end

if createFigures
    figure
    %set(gcf, 'visible','off');
    bar(classP);
    title(strcat('Classification Results: Part ', int2str(part)));
    xlabel('Clip to classify');
    ylabel('Decision variable');
    %saveas(gcf, strcat('part', int2str(part), 'class'),'png');

    figure
    %set(gcf, 'visible','off');
    plot(diag(s), 'ko');
    title(strcat('Singular Values: Part ', int2str(part)));
    xlabel('Number of features');
    ylabel('Value of feature');
    %saveas(gcf, strcat('part', int2str(part), 'sig'),'png');
end

if printProcessing
    fprintf('Success at Part %d\n', part);
    fprintf('Training trials = %d\n', trainTrials);
    fprintf('Testing trials = %d\n', testTrials);
    fprintf('Accuracy using projected vectors %0.2f\n', mean(accuracyP));
end
```

```
end

function [dat, mincols] = getSpec(threeDirs, clipNum, printProcessing, ...
    oldmincols)
%ensure mincols accross different trials

clipLen = 3; % seconds
sampleFactor = 1;

filterType = 'gauss';
bandpass = 0;
bpWidth = 0; bpCenter = 0;
amp = 1;
width = 2000;
samples = 5;

previousNum = 9999;
for k = 1:length(threeDirs)
    cd (threeDirs{k}); % go to specified dir for song count
    filelist = dir('*.wav');
    cd ..;  % return to main dir
    songNum = length(filelist);
    previousNum = min([songNum, previousNum]);
    cols = 0;
    dat.(threeDirs{k}) = [];
    for song = 1:previousNum;
        if printProcessing
            fprintf(strcat('Processing song:\t',...
                    filelist(song).name, '...\n'));
        end
        cd (threeDirs{k}); % go to specified dir for song
        [temp, fs] = wavread(filelist(song).name);
        cd ..;  % return to main dir
        % average both channels
        temp = mean(temp,2); % average two channels
        vec = temp;
        vec = resample(temp, 1, sampleFactor);
        clear temp;
        fs = fs / sampleFactor;
        len = length(vec);
        % compute last possible clipping location in samples
        last = floor(len - 1.5 * (clipLen * fs));
```

```
        % store trainTrials random valid indexes to get clip from
        rnc = randi([1, last],1, clipNum);
        for l = 1:clipNum
            soundVector = trimSoundVector(vec, fs, clipLen, rnc(l),1,1);
            windowLength = floor(length(soundVector) / samples);
            noverlap = floor(.6 * windowLength);
            [spec, f, t] = spectrogram(soundVector, windowLength,...
                noverlap, 'onesided', fs);
            % one col per trial
            spec = abs(spec);
            specCol = reshape(spec, prod(size(spec)), 1);
            % stack each by col
            dat.(threeDirs{k}) = [ dat.(threeDirs{k}), specCol ];
            cols = cols + 1;
        end
    end
    assert((cols == (previousNum * clipNum)), 'cols does not match');
end
mincols = previousNum * clipNum;
end

function [accuracyP, classP ] = classifySVD(u, s, v, ...
    trainTrials, testTrials, mincols, threeDirs, features);

group = ones(trainTrials, 1);
group = [group; 2*group; 3*group];
temp = ones(testTrials, 1);
trueComparison = [ temp; 2*temp; 3*temp ];
trainM = [];
trainP = [];
testM = [];
testP = [];

featuresProj = 1:features; % modes to use
featuresMode = 1:features; % modes to use
weighted = s*v';
v = weighted';
startInd = 1;

% for each type get the corresponding components in u or v
% concatenate all types
for k = 1:length(threeDirs)
```

```
    % by projection vectors
    projections.(threeDirs{k}) = v(startInd:(startInd + mincols - 1), ...
        featuresProj);
    % of those components choose a subset for training
    trainP = [trainP; projections.(threeDirs{k})(1:trainTrials, :)];
    % use the rest for testing
    testP = [testP; ...
        projections.(threeDirs{k})(((trainTrials + 1):end), :)];

    % '' by mode vectors
    modes.(threeDirs{k}) = u(startInd:(startInd + mincols - 1), ...
        featuresMode);
    trainM = [trainM; projections.(threeDirs{k})(1:trainTrials, :)];
    testM = [testM; ...
        projections.(threeDirs{k})((trainTrials + 1):end, :)];

    startInd= startInd + mincols;
end

% % verify
% [m,n] = size(testP);
% assert(m == (length(featuresProj) * length(threeDirs)), 'improper test trials');
% [m,n] = size(testM);
% assert(m == (length(featuresMode) * length(threeDirs)), 'improper test trials');
% [m,n] = size(trainP);
% assert(m == (length(featuresProj) * length(threeDirs)), 'improper train trials');
% [m,n] = size(trainM);
% assert(m == (length(featuresMode) * length(threeDirs)), 'improper train trials');

% must have the same cols, rows are the trials
% classify by projections
[classP, errP] = classify(testP, trainP, group);
% classify by modes
[classM, errM] = classify(testM, trainM, group);

% cross validation
accuracyP = length(find(classP == trueComparison)) / trainTrials;
accuracyM = length(find(classM == trueComparison)) / trainTrials;

function [output] = trimSoundVector(input, fs, new_length, startInd, gate_start, gate_en
% trims a vector by the length in seconds specified by new_length
% (starting from 'start') and gates the trimmed ends to prevent popping
```

```
% takes new_length in seconds
% assumes a vector is passed

[m,n] = size(input);
assert(n == 1, 'trimSoundVector must be passed single column vector');

new_length = floor(new_length * fs); % convert to samples

%cut
input = input(startInd:end);

[rows, cols] = size(input);
if rows < new_length
    temp = zeros(new_length, 1); % add zeros to the end
    for k = 1:rows
        temp(k) = input(k);
    end
    output = temp;
else
    output = input(1:new_length); % truncate vector at row new_length
end

output = createGate(output, fs, gate_start, gate_end);
[rows, cols] = size(output);
assert((rows == new_length), 'error in trimletters: not all letters equal to new_length'
end
```

References

[1] Kirk Baker. Singular Value Decomposition Tutorial. pages 14–21, 2005.

[2] J N Kutz. *Data-Driven Modeling & Scientific Computation: Methods for Complex Systems & Big Data*. OUP Oxford, 2013.