

# Landmark Classification: PCA, LDA, and Support Vector Machines

CLSP Summer Workshop 2004

June 30, 2004

## 1 Spectrograms and Problem Definition

The directory `landmark_waves` contains a number of waveform files. Each waveform file is a 150ms snippet, excised from a longer sentence, so that the midpoint is a landmark (a consonant closure or release, for consonants that are nasals, stops, or fricatives). The waveforms are stored in subdirectories of the form `landmark_waves/${lm}`, where `${lm}` is a landmark label. A landmark label is either `+${ph}` or `${ph}+`, representing closures and releases, respectively, of the phoneme `${ph}`.

Choose a distinctive feature of interest to you. You may choose one of the features given in Table 1, or you may choose any other binary division of the phonemes that seems likely, to you, to result in good classification performance. Use the `wavread` command in matlab<sup>1</sup> to load several examples of `[-feature]` landmark waveforms, and several examples of `[+feature]` waveforms, for your chosen `feature`.

Make sure that you have the `voicebox` toolkit in your matlab search path; you can set the search path using the `path` command. Plot spectrograms of each waveform with a 500Hz analysis bandwidth, using the `voicebox spgrambw` function, i.e., `spgrambw(WAV,8000,500)`. Look at the `[+feature]` waveforms. Now look at the `[-feature]` waveforms. Are there any consistent differences? Consider, in particular, the formant frequencies, the burst spectrum of stops, and the frication spectrum of fricatives. If you are interested, there is a table, in Appendix A, of the most widely attested acoustic correlates of distinctive features.

A complete linear-frequency spectrogram, as computed by `spgrambw`, is usually too much data for statistical analysis. The data size can be reduced slightly, without too much loss of distinctive feature information, by creating a mel-scale spectrogram, using the code snippet shown in fig. 1. Notice that relatively long code snippets of this sort may be stored in text files called scripts and functions, so that you don't need to retype them over and over again: see Appendix B.9 for more information. Create mel-scale spectrograms of several `[+feature]` and several `[-feature]` waveforms, and plot the results using `imagesc`. Label the abscissa in milliseconds, and the ordinate in Hertz, as shown in Fig. 1. Note: matlab 6.5.0 has a bug that causes `imagesc` to ignore a nonlinear frequency axis, such as that in the vector `FREQS`. If your version of matlab has this bug, use the last five lines of code in Fig. 1 to correctly label the frequency axis in Hertz.

Look for the feature-specific acoustic correlates that you spotted when using a 2ms, 512-point spectrogram. Are the same acoustic distinctions still visible in the mel-scale spectrogram? If not, consider using a mel filter bank with more bands, or use a shorter frame skip length.

## 2 Vectorizing the data

Vectorize one of the mel-scale spectrograms you created in part 1. In matlab, a matrix `MELGRAM` can be vectorized using the notation `svect=MELGRAM(:);`. Use `[NBANDS,NFRAMES]=size(MELGRAM)` to compute the size of the spectrogram matrix. Use `size(svect)` to compute the size of the vectorized spectrogram.

Unfold the vector back into a matrix. One way to do this, in matlab, is as follows: `S=zeros(NBANDS,NFRAMES); S(:)=svect;` Use `imagesc` to plot the unfolded spectrogram. Make sure that it is identical to the spectrogram you started with.

Load about 1000 waveforms—500 `[+feature]` waveforms, 500 `[-feature]` waveforms, with roughly equal representation from at least two different `[+feature]` phonemes and at least two different `[-feature]`

---

<sup>1</sup>Before you use any new matlab command, it is strongly recommended that you read the help page describing command syntax: for example, you can type `help wavread` to read about `wavread`.

phone	sonorant	continuant	lips	blade	body	anterior	strident	voiced
b	-	-	+	-	-	+		+
d	-	-	-	+	-	+		+
g	-	-	-	-	+	-		+
p	-	-	+	-	-	+		-
t	-	-	-	+	-	+		-
k	-	-	-	-	+	-		-
m	+	-	+	-	-	+		
n	+	-	-	+	-	+		
ng	+	-	-	-	+	-		
f	-	+	+	-	-	+	-	-
th	-	+	-	+	-	+	-	-
s	-	+	-	+	-	+	+	-
sh	-	+	-	+	-	-	+	-
v	-	+	+	-	-	+	-	+
dh	-	+	-	+	-	+	-	+
z	-	+	-	+	-	+	+	+
zh	-	+	-	+	-	-	+	+

Table 1: Distinctive feature notation for the consonants of English, based on the book *Acoustic Phonetics* by Ken Stevens. Feature “strident” is defined only for fricatives, and feature “voiced” is undefined for nasals. Features “blade” and “body” are redundant, but may be used to identify errors in the outputs of the other classifiers.

```

% Create 32 mel-scale filterbanks, for use on a 512-point FFT
W=melbankm(32,512,8000);
% Cut WAV into 160-sample windows, overlapping by 120 samples
FRAMES=enframe(WAV,160,40);
% Compute magnitude STFT
MSTFT=abs(fft(FRAMES,512,2));
% Multiply MSTFT times W to create mel-scale spectrogram
MELGRAM=20*log10(W*MSTFT(:,1:257)');
% Compute center frequencies, in Hertz, of each filter
FREQS=round(mel2frq([1:32]*frq2mel(4000)/33));
% Compute time alignments, in milliseconds, of each frame
TIMES=[-140:5:140];
% Create an image plot of the mel-scale spectrogram
imagesc(TIMES,FREQS,MELGRAM);
% Flip the frequency axis, so low frequency is at bottom
axis xy;
%% Alternate code -- necessary only if your version of matlab has
%% the bug that causes nonlinear Y-axis to fail
imagesc(TIMES,[1:32],MELGRAM); axis xy;
YTick=get(gca,'YTick');
YTickLabel='';
for I=YTick, YTickLabel=strvcat(YTickLabel,sprintf('%d',FREQS(I))); end
set(gca,'YTickLabel',YTickLabel);

```

Figure 1: Matlab code snippet: creating and plotting a mel-frequency spectrogram.

```

% List of +feature release landmarks
% -- change this to suit the feature that you're using
% -- change this if you're using closures instead of releases
PLUSPHONES={'b+', 'p+', 'm+', 'f+', 'v+'};
% Get directory listings of all directories given by PLUSPHONES
ROOTDIR='/export/ws04ldmk/tutorial/landmark_waves/';
for I=1:length(PLUSPHONES), PLUSDIRS{I}=dir([ROOTDIR, PLUSPHONES{I}]); end
% Load odd-numbered waves to TRAIN, even-numbered waves to TEST
for I=0:499,
    % FILE_NUM and DIR_NUM are ratio and remainder of I/length(PLUSPHONES)
    FILE_NUM=3+floor(I/length(PLUSPHONES));
    DIR_NUM=1+rem(I,length(PLUSPHONES));
    % Load the waveform file
    WAV=wavread([ROOTDIR,PLUSPHONES{DIR_NUM}, '/', PLUSDIRS{DIR_NUM}(FILE_NUM).name]);
    % Convert to mel-scale spectrogram, and load it to TRAIN
    MSTFT=abs(fft(enframe(WAV,160,40),512,2));
    MELGRAM=20*log10(W*MSTFT(:,1:257)');
    TRAIN(I+1,:) = MELGRAM(:)';
end

```

Figure 2: Matlab code snippet: A method for loading 500 waveform files into the TRAIN array.

phonemes. You should either choose to focus on closure landmarks or release landmarks, but not both. One method for efficiently loading 500 waveforms is shown in Fig. 2. Convert the waveforms into mel-scale spectrograms, vectorize them, and stack them into a single matrix called something like TRAIN. From a different list of 1000 waveforms (possibly the next 500 in PLUSDIRS and MINUSDIRS), load vectorized mel-scale spectrograms into a matrix called TEST.

Normalize both data matrices to have zero mean and unit standard deviation, as shown here:

```
X_TRAIN=(TRAIN-repmat(mean(TRAIN),[1000 1]))./repmat(std(TRAIN),[1000 1]);
```

Verify that you can reconstruct a mel-scale spectrogram from any row of the normalized data matrices X\_TRAIN and X\_TEST. Use subplot and imagesc to plot mel-scale spectrograms corresponding to the first two [+feature] data vectors, and corresponding to the first two [-feature] data vectors. Always label the abscissa in milliseconds, and the ordinate in Hertz, as shown in Fig. 1.

### 3 PCA

Use  $\Sigma = \text{cov}(X\_TRAIN)$  to compute the global covariance matrix of the normalized training data. The principal components are the eigenvectors of  $\Sigma$ , that is, they are the vectors  $\vec{v}_i$  such that, for some scalar  $d_i$ ,

$$\Sigma \vec{v}_i = d_i \vec{v}_i \quad (1)$$

In matlab, the eigenvectors can be efficiently computed using the eig function.  $[PCA\_V, PCA\_D] = \text{eig}(\Sigma)$  returns matrices such that  $PCA\_V(:, i) = \vec{v}_i$ ,  $PCA\_D(i, i) = d_i$ ,  $d_i > d_{i-1}$ ,  $V^{-1} = V'$ , and  $V' * V = I$  where  $I$  is the identity matrix.

Assume that the rows of X\_TRAIN are data vectors, and the columns of PCA\_V contain principal components; then the rows of X\_TRAIN can be rotated into the new PCA space using  $PCA\_TRAIN = X\_TRAIN * PCA\_V$ . Transforming the data in this way decorrelates the columns of PCA\_TRAIN:  $\text{cov}(PCA\_TRAIN)$  should equal the matrix PCA\_D. In order to recover the original data, you can post-multiply by the inverse transform matrix:  $PCA\_TRAIN * PCA\_V'$  should equal X\_TRAIN.

Create the transformed data matrices PCA\_TRAIN and PCA\_TEST.

Use plot to create a two-dimensional scatter plot of a few of the transformed vectors, e.g.,

```

[M,K]=size(PCA_TRAIN);
hold off;
plot(PCA_TRAIN(1:100,K),PCA_TRAIN(1:100,K-1),'r+');
hold on;
plot(PCA_TRAIN(501:600,K),PCA_TRAIN(501:600,K-1),'bo');

```

Choose the two dimensions, from matrix PCA\_TRAIN, with the highest variance (i.e., the highest corresponding elements of PCA\_D). Plot about 100 [+feature] vectors, and about 100 [-feature] vectors. Repeat with the next two dimensions; you should have a total of two scatter plots, showing the four principal components with largest variance.

Another way to interpret the principal components is as follows: the measurement PCA\_TRAIN(i,k) measures the extent to which vector X\_TRAIN(i,:) is similar to the “spectrogram shape” represented by vector PCA\_V(:,k). Thus if PCA\_D(K,K) is the largest variance, then PCA\_V(:,K) is the shape vector that explains the most variance in the training data.

Unfold PCA\_V(:,K) into a mel-scale spectrogram, and plot it using `imagesc`. Label the abscissa in milliseconds, and the ordinate in Hertz. What shape of spectrogram has the highest variance?

Repeat using all four of the principal components with highest variance.

## 4 LDA

In this section, you will compute three different types of feature projections, and associated classifiers, using linear discriminant analysis.

The linear discriminant vector is based on the mean-difference vectors,  $\vec{v}_{\pm} = \vec{\mu}_{+} - \vec{\mu}_{-}$ , where  $\vec{\mu}_{+}$  is the mean of all data vectors in class [+feature], i.e., `MU_PLUS=mean(X_TRAIN(1:500,:))'`. The vector  $\vec{v}_{\pm}$  is already a pretty good classifier: a particular data vector  $\vec{x}_m$  is considered to have the label [+feature] if and only if

$$\vec{x}_m' \vec{v}_{\pm} > b_{\pm} \quad (2)$$

where, if  $\vec{x}_m$  has zero mean, and if there are equal numbers of positive and negative examples, the threshold  $b_{\pm}$  is guaranteed to be zero. Try this classifier: compute the mean difference vector V\_MUDIFF, and transform all of your training and test data vectors to create vectors of scalar discriminant functions MUDIFF\_TRAIN=X\_TRAIN\*V\_MUDIFF and MUDIFF\_TEST. Plot histograms of the [+feature] and [-feature] tokens of MUDIFF\_TRAIN, using code such as

```

BINS=min(MUDIFF_TRAIN)+[0:0.05:1]*(max(MUDIFF_TRAIN)-min(MUDIFF_TRAIN));
NPLUS=hist(MUDIFF_TRAIN(1:500),BINS);
NMINUS=hist(MUDIFF_TRAIN(501:600),BINS);
plot(BINS,NPLUS,'r',BINS,NMINUS,'b');

```

Now do the same for MUDIFF\_TEST.

Compute the classification error rate of this classifier on the training corpus. Assuming that the classification threshold is zero, the number of classification errors is given by the histogram as

$$\text{ERR}=\text{sum}(\text{NPLUS}(\text{BINS}>0))+\text{sum}(\text{NMINUS}(\text{BINS}<=0));$$

What is the classification error rate of this classifier on the training corpus? On the test corpus? Note that if the training corpus error is greater than 50%, you should probably swap the less-than and greater-than signs.

Convert the vector  $\vec{v}_{\pm}$  into a spectrogram, and plot it using `imagesc`. What time-frequency pixels are more heavily associated with [+feature] tokens? Which pixels are more heavily associated with [-feature] tokens?

The classical two-class linear discriminant transform vector is

```
V_LDA = inv(Sigma)*V_MUDIFF;
```

where  $\Sigma$  is the global covariance matrix (the covariance of all tokens—the same matrix that you used to compute PCA). Compute  $\vec{v}_{LDA}$ . Transform the data, to create discriminant vectors `LDA_TRAIN=X_TRAIN*V_LDA` and `LDA_TEST`. Plot histograms of the `[+feature]` and `[-feature]` tokens of `LDA_TRAIN`. Do the same with `LDA_TEST`. What is the training corpus error rate of this classifier? What is the test corpus error rate? Why are they different? Convert  $\vec{v}_{LDA}$  into a spectrogram matrix, and plot it using `imagesc`. What happened?

Quite probably, you have just observed what happens when a classifier gets over-trained. The only way to avoid over-training a classifier is by controlling its Vapnik-Chervonenkis (VC) dimension. The classical way of controlling the VC dimension of a classifier is by reducing the number of trainable parameters. Try reducing the dimensionality of your LDA classifier using principal components analysis, as follows: (1) transform the training data using PCA, (2) extract the 25 dimensions (or so) with highest variance, (3) perform a 25-dimensional LDA on the extracted data, resulting in a short-LDA transform vector  $\vec{v}_{SLDA}$ , (4) use  $\vec{v}_{SLDA}$  to transform the test data, (5) compute histograms of the training and test data discriminants. What is the training corpus error rate? What is the test corpus error rate? You should find that SLDA gives much lower generalization error (defined as the difference between test corpus error and training corpus error) than does classical LDA. Apply inverse-PCA to  $\vec{v}_{SLDA}$  in order to get back a full spectrogram matrix, and plot it using `imagesc`. What time-frequency pixels are attributed, by the SLDA classifier, to `[+feature]`? What pixels are attributed to `[-feature]`?

## 5 SVM

In this section, you will train a linear support vector machine.

A linear support vector machine is a classifier vector,  $\vec{v}_{SVM}$ , created as a weighted sum of all of the tokens in the training database. Let  $X$  be an  $M \times K$  matrix whose rows are the training vectors  $\vec{x}'_m$ ,  $1 \leq m \leq M$ . Let  $\vec{y} = [y_1, \dots, y_M]'$  be a vector whose element  $y_m$  is the correct label of vector  $\vec{x}_m$ , i.e. either  $y_m = 1$  or  $y_m = -1$ . Then

$$\vec{v}_{SVM} = X\vec{\alpha} \quad (3)$$

where  $\vec{\alpha}$  is an  $M$ -vector of weights, computed by minimizing

$$\mathcal{E} = \arg \min \vec{\alpha}'(XX')\vec{\alpha} - \vec{\alpha}'\vec{y} \quad (4)$$

subject to the following two constraints: first,  $\vec{\alpha}$  must be a zero-mean vector, i.e.,

$$\sum_m \alpha_m = 0 \quad (5)$$

and second, each weight  $\alpha_k$  must have the same sign as the corresponding label, i.e.

$$0 \leq y_k \alpha_k \leq C \quad (6)$$

where  $C$  is an arbitrary parameter specifying the degree to which the algorithm should pay attention to outliers (data vectors that are very far on the wrong side of the classification boundary). Equation 4 is the key criterion to be minimized. Notice that it has two terms. The first term is related to the VC dimension of the classifier; the second term is related to the training corpus error. By minimizing the sum of these two terms, it's possible to simultaneously control the training corpus error *and* the generalization error of the classifier.

The problem of minimizing Eq. 4, subject to the constraints in Eqs. 5 and Eq. 6, is an example of a “quadratic programming” problem, or QP. Programming QP yourself in matlab is possible but very slow; it is easier to use an existing efficient QP solver. An example of an efficient QP-solving program is the `svm_learn` program, available in `/apps/svmlight`. In order to use `svm_learn`, you need to output your data vectors in a text file with the following format:

```
label1 index1:val1 index2:val2 ...
```

Sample code snippets for reading and writing data in SVM-file format are given in Fig 3.

Output your entire 2000-token training dataset in the format required by `svm_learn`. Use `svm_learn data1.txt linear.svm` to train a linear SVM from `data1.txt`, and store the result in `linear.svm`. After

```

% Create the training corpus label vector
Y = [repmat(-1,[1 500]), repmat(1,[1 500])]';
% Create the format vector for writing SVM data
FMT=['%f',repmat(' %d:%f',[1 size(X_TRAIN,2)]), '\n'];
% Open an output file, and write out data vectors using column-order fprintf
fid=fopen('data1.txt','w');
for I=1:1000,
    fprintf(fid,FMT, Y(I), [1:size(X_TRAIN,2); X_TRAIN(I,:)]);
end
fclose(fid);
% Read input from support_vectors.txt, delimited by colon or space
A = textread('support_vectors.txt','','whitespace',' :\b\t');
alpha=A(:,1);
support_vectors=A(:,3:2:size(A,2));

```

Figure 3: Matlab code snippet: writing data vectors and reading the classifier definition in `svm_learn` data format. Note: use the `tail` program or a text editor get rid of the first eleven lines of the SVM input file before using this code.

training, the first eleven lines of `linear.svm` will contain classifier statistics, including the classification threshold. Each line after the tenth contains one scalar weight  $\alpha_m$ , followed by all of the elements of the corresponding support vector  $\vec{x}_m$ , stored in “index:value” format. Use a text editor, or the `tail` program in unix, to clip the first eleven lines from this file, and store the rest in a file `support_vectors.txt`. Read `support_vectors.txt` (e.g., using the code snippet in 3). Construct the SVM discriminant vector  $\vec{v}_{SVM}$ , as shown in Eq 3. Transform the training and test data in order to create discriminant vectors `SVM_TRAIN=X_TRAIN*V_SVM` and `SVM_TEST`. Plot histograms of both `[+feature]` and `[-feature]` tokens, for both the training and test data. What is the training corpus error rate (be careful to use the correct classification threshold, given on line 11 of the SVM file, don’t just use zero as a threshold)? What is the test corpus error rate? Plot  $\vec{v}_{SVM}$  as a spectrogram. What time-frequency pixels are associated with each class?

If you’re interested, try using `svm_learn` to learn a nonlinear (RBF) support vector machine. Try using `svm_classify` to classify your test data with the new classifier. Can you get classification results that are better than those of the linear classifier?

## 6 Human Speech Recognition Performance

Load about 50 `[+feature]` waveforms, and about 50 `[-feature]` waveforms, using code similar to that given in Fig. 2, but without converting the waveforms into mel-frequency spectrograms (i.e., keep the data as waveforms). Randomly reshuffle the 100 waveforms. Keep a record of the shuffling order, but do not look at it yet. For example,

```

[foo, SORT_ORDER]=sort(rand([1 100]));
for tok_index=1:100,
    SHUFFLED_WAVS(:,tok_index)=WAVEFORMS(:,SORT_ORDER(tok_index));
end

```

Put on a pair of headphones, in order to make sure that you can clearly hear the audio. Play each of the shuffled waveforms, in order (you may find it convenient to have matlab step through them for you, using the code in Fig. 4). Listen to each waveform as many times as you like. Write down the phoneme label that you think you hear.

Now, look at the `SORT_ORDER` matrix, and use it to score your speech perception results. Compute your classification error rate for the binary distinctive feature that you’ve been investigating in previous parts of this lab. Is your brain a better phoneme classifier than the SVM? How about the LDA?

```

LABELS=cell(100,1);
for I=1:100,
    while(length(LABELS{I})<1),
        soundsc(SHUFFLED_WAVS(:,I));
        LABELS{I}=input(sprintf('Enter a phoneme (or hit return to hear wave %d again): ',I),'s');
    end
end

```

Figure 4: Matlab code snippet: Play each waveform, then ask for a phoneme label. If user types a phoneme, enter it in LABELS, and proceed to the next waveform; if not, play the same waveform over again.

If your error rate is non-zero, create a confusion matrix. Does your confusion matrix indicate that some manners or places of articulation are more confusable than others? Is there a bias in favor of certain manners or places of articulation?

## 7 Conclusion

This section ends with some topics for discussion. You are not required to answer these questions, but you should think about them.

Under what circumstances do classical statistical methods, such as linear discriminant analysis, fail? Why is it possible for regularized learners, such as the support vector machine, to succeed under similar circumstances?

A mixture Gaussian classifier is a much better classifier than LDA, but it is susceptible to over-training in exactly the same way that LDA is susceptible to over-training. Nevertheless, for the past 20 years, automatic speech recognition systems have used mixture Gaussian classifiers, and have only occasionally run into over-training problems. Why?

Did the human listener achieve zero error on the task of distinctive feature classification in this lab? If not, is there anything that could be done (to the data, or to the listener) to improve performance?

Did any of the automatic classifiers achieve zero error on this task? If not, what could be done to improve performance?

All experiments in this lab involved binary classification of fixed-length speech waveforms. How can the methods in this lab be extended to cope with variable-length waveforms and variable-length label strings?

## Appendix A Acoustic Correlates of Distinctive Features

“Acoustic correlates” of a distinctive feature are acoustic measurements that can be used to determine whether a phoneme is [+feature] or [-feature]. Phoneticians are divided into two camps: (1) those who believe that each distinctive feature is primarily cued by pinpoint measurements at particular times, in particular frequency bands (Blumstein and Stevens, JASA, 1979), and (2) those who believe that every possible acoustic measurement carries information about every possible distinctive feature (Kewley-Port, JASA, 1982). The engineering response to this debate is, as always, to conclude that both camps are correct: every measurement informs us about almost every distinctive feature, but some measurements are more informative than others (in very precise terms: some measurements have higher mutual information with the target distinctive feature than others).

Table 2 lists some widely attested acoustic correlates. With some practice, you can use this table to teach yourself spectrogram reading. These acoustic correlates can also be useful in automatic speech recognition, but they should usually augment the mel-scale spectral observation vector, not replace it. In this table, a “formant” is a resonant frequency of the vocal tract; it shows up in the spectrogram as a thick fuzzy bar, like a horizontal caterpillar. During most vowels,  $250 \leq F_1 \leq 1000\text{Hz}$ ,  $900 \leq F_2 \leq 2400\text{Hz}$ ,  $2200 \leq F_3 \leq 3000\text{Hz}$ ;  $F_3$  may or may not be observable on spectrograms computed from telephone speech. The “formant locus” is the frequency that the formant would take right at the instant of consonant closure or release, *if* you could actually measure the formant at that time—often it is impossible to actually measure the formant at

that time, so you must interpolate the formant backward in time from a following vowel, or forward in time from a preceding vowel. All English consonants have an F1 locus of about 200Hz. The F2 and F3 loci are variable but useful cues for place of articulation (lips vs. blade vs. body). Other place cues include stop burst spectrum (the spectrum of the noise that occurs at  $t = 0$ ) and frication spectrum (the spectrum of noise during the closed portion of a fricative). Stop consonant voicing, in English, is primarily cued by voice onset time (VOT), the time delay between the burst and the onset of vowel voicing.

## Appendix B Matlab Reference

The following sections give some useful reference material. They borrow heavily from the document “Using matlab on athena,” <http://web.mit.edu/olh/Matlab/Matlab.html>.

### Appendix B.1 Getting Help on Matlab

demo	run a demo
helpdesk	start a graphical “help desk” with help and tutorial menus
help	get a list of available help topics
help [function]	get syntax and help for [function]
which [command]	see the full path of [command]
which [command] -all	list other versions of [command]
what [directory]	see what commands are in [directory]
lookfor [keyword]	find a command by keywords

### Appendix B.2 Creating Matrices

The first four examples below shows how to create a row vector and a column vector. The remaining examples show how to work with matrices. The last two examples show how to treat a matrix as if it were a vector (stacked in column order).

```
>> x = [1;2;3;4];
x =
     1
     2
     3
     4
>> x = [1 2 3 4];
x =
     1     2     3     4
>> y = x';
y =
     1
     2
     3
     4
>> x(3) = 5
x =
     1     2     5     4
>> x = [1 2 3 4; 0 9 3 8]
x =
     1     2     3     4
     0     9     3     8
>> x = [4+5i 2 4; 1 3+i 7]
x =
 4.0000 + 5.0000i  2.0000  4.0000
```

FEATURE	CONTEXT	INFORMATIVE ACOUSTIC CORRELATES
sonorant	all	strong periodic voicing, with a total energy that doesn't change much from frame to frame during consonant closure, and with a spectral peak during closure between 250Hz and 1000Hz
continuant	all	high-frequency energy during closure (above 1000Hz) is not more than 30dB below the low-frequency energy during closure, and high-frequency energy is therefore visible in the spectrogram
lips	any  fricative  stop release  stop release	F2 and F3 formant loci are lower than the formant frequencies of any preceding or following vowel, thus formants rise into a following vowel, fall from a preceding vowel frication spectrum (during closure) has very low amplitude, with roughly equal energy at all frequencies above 1000Hz burst spectrum (at t=0ms) has very low amplitude, with roughly equal energy at all frequencies above 1000Hz VOT is shorter than predicted by voicing features
blade	any stop release  fricative	formant loci are $1600 < F_2 < 2000\text{Hz}$ , $F_3 \approx 3000\text{Hz}$ burst spectrum is low amplitude and highpass, with more energy at high frequencies (above 2500Hz) than at low frequencies (below 2000Hz) frication spectrum is low-amplitude and highpass
body	any  stop release  fricative	formant loci may be at any frequency at all, but F2 locus and F3 locus are always very close together burst spectrum shows a compact peak (on the spectrogram, a lump of visible energy) between 1000Hz and 3500Hz frication spectrum shows a compact peak
anterior	any  stop or fricative	<b>[-anterior]</b> phones have F2 and F3 formant loci close together, <b>[+anterior]</b> phones have F2 and F3 loci far apart <b>[-anterior]</b> phones have a compact frication or burst peak between 1000Hz and 3500Hz, <b>[+anterior]</b> phones have a diffuse spectrum with no obvious peak
strident	fricative	high-frequency energy (above 3000Hz) is as strong during the consonant as it is during following and preceding vowels
voiced	any any closure   fricative release  stop release	<b>[+voiced]</b> consonants are shorter than <b>[-voiced]</b> consonants all stops and fricatives (both voiced <i>and unvoiced</i> ) tend to have a "voice bar:" a few extra frames, after oral closure, with periodic voiced energy continuing in the very-low-frequency range (below 300Hz). Voiced stops and fricatives tend to have a longer voice bar than unvoiced ones voiced fricatives, but <i>not stops</i> , may have a voice bar for a few frames before release unvoiced stops have a VOT longer than 25ms

Table 2: Some of the most widely attested acoustic correlates of distinctive features.

```

    1.0000          3.0000 + 1.0000i    7.0000
>> x(2,2) = 3
x =
    4.0000 + 5.0000i    2.0000          4.0000
    1.0000          3.0000          7.0000
>> x(2,2:3) = [1 5]
x =
    4.0000 + 5.0000i    2.0000          4.0000
    1.0000          1.0000          5.0000
>> x(:,1) = [6;7]
x =
    6.0000          2.0000          4.0000
    7.0000          1.0000          5.0000
>> y = x(2,:)
y =
    7.0000          1.0000          5.0000
>> y = x(:)
y =
    6.0000
    7.0000
    2.0000
    1.0000
    4.0000
    5.0000
>> x(3) = 16
x =
    6.0000          16.0000          4.0000
    7.0000          1.0000          5.0000

```

You can also specify a range J:D:K, that steps from J to K in increments of D. For example:

```

>> z = (1:3:7)
z =
     1     4     7
>> z = (14:-2:5)
z =
    14    12    10     8     6

```

Matrix creation functions:

ones([m n])	create a matrix of all ones
zeros([m n])	create a matrix of all zeros
rand([m n])	uniformly distributed random numbers between 0 and 1
randn([m n])	unit variance Gaussian random numbers
eye(n)	$n \times n$ identity matrix
magic(n)	$n \times n$ magic square

Useful special constants (if you redefine one of these, your new definition will replace the original definition):

```

i    sqrt(-1)
j    sqrt(-1)
pi   pi

```

Matrix property functions:

rank(M)	1 if M is vector, 2 if matrix, 3 if rank-3 tensor, etc.
size(M)	return a row vector specifying the size of matrix M
size(M,r)	return the size of M along rank r
length(M)	equivalent to max(size(M))

### Appendix B.3 Saving Your Work

save filename	save all current variables into a file called filename.mat
load filename	load all variables from filename.mat
diary filename	record everything you type in filename
diary off	turn off diary recording
fopen	open a filename for binary or text output or input
fwrite	binary output
fread	binary input
fprintf	pretty-print text output with a given format
textread	read text input with a given format

For example, suppose that the file `foo.svm` contains more than 1000 lines, each of the form

```
weight ind1:val1 ind2:val2 .... ind64:val64
```

`foo.svm` can be read as

```
>> A = textread('foo.svm',' ','whitespace',' :\b\t');
>> weights = A(:,1);
>> indices = A(:,2:2:size(A,2));
>> values = A(:,3:2:size(A,2));
```

### Appendix B.4 Basic Arithmetic

The basic operators are

'	matrix transpose
+	addition
-	subtraction
*	matrix multiplication
/	matrix division
^	matrix exponentiation
.*	element-wise multiplication
./	element-wise division
.^	element-wise exponentiation

```
>> x = [1 2 3];
>> y = [5 6 2];
>> w = x + 2
w =
     3     4     5
>> w = x+y
w =
     6     8     5
>> w = x .* y
w =
     5    12     6
>> w = x * y
Error: dimensions must match
>> w = x' * y
```

```

w =
    23
>> w = x .^ 2
w =
    1    4    9
>> w = y .^ x
w =
    5    36    8
>> w = y ./ x
w =
    5.0000    3.0000    0.6667
>> x = [2 -4 3-4i -3i];
>> y = abs(x)
y =
    2    4    5    3
>> phase = angle(x)
phase=
0  -3.1416  -0.9273  -1.5708
>> x = [4 -9 i 2-2i];
>> y = sqrt(x)
y =
    2.0000    0 + 3.0000i    0.7071 + 0.7071i    1.5538 - 0.6436i

```

The following functions act element-wise on a matrix: the output of the function has the same size as the input.

abs	absolute value
real	real part of a complex number
imag	imaginary part of a complex number
angle	phase of a complex number
conj	complex conjugate
sqrt	square root
pow	raise to an arbitrary power
exp	raise $e$ to the power of each element
log	natural logarithm
log10	base-10 logarithm
sin,cos,tan	trigonometric functions (argument in radians)
sinh,cosh,tanh	hyperbolic functions
asin,acos,atan	inverse trigonometric functions
atan2	two-argument arctangent function, returns angle between $-\pi$ and $\pi$
round	round to the nearest integer
fix	round to the nearest integer toward zero
floor	round to nearest integer toward negative infinity
ceil	round to nearest integer toward positive infinity

If  $\text{rank}(M)=1$ , then  $\text{max}(M)$  will find the maximum element of the vector  $M$ . If  $\text{rank}(M)=2$ , then  $\text{max}(M)$  returns a row vector showing the maximum element in each column of  $M$ . In order to get the maximum in each row, use  $\text{max}(M, [], 2)$ . In order to get the maximum element in the whole matrix, use  $\text{max}(\text{max}(M))$  or  $\text{max}(M(:))$ . Functions with similar behavior include:

$\text{max}(M)$	maximum
$\text{min}(M)$	minimum
$\text{sum}(M)$	sum
$\text{prod}(M)$	product

The  $\text{cumsum}$  and  $\text{cumprod}$  functions return cumulative sum or product of a vector, or of the columns of a matrix. For example:

```
>> x = [1 3 1];
>> y = cumsum(x)
y =
    1     4     5
```

## Appendix B.5 Speech Signal Processing

The function `conv` convolves its inputs. For example

```
>> x = [1 3 5 4 2];
>> y = [1 1];
>> z = conv(x,y)
z =
    1     4     8     9     6     2
```

The function `conv2` implements 2D (matrix) convolution. `filter` and `filter2` implement filtering with arbitrary numerator and denominator polynomials. The function `diff(x)` computes the differences between consecutive elements of `x`; it is equivalent to `conv(x,[1 -1])`.

`fft(x)`, if `x` is a vector, computes its discrete Fourier transform. `fft(M)`, if `M` is a matrix, computes the DFT separately of each column of `M`. `fft(M,[],2)` computes the DFT of each row of `M`.

The `voicebox` toolkit includes a number of useful signal processing functions, including `frq2mel` and `mel2frq` (convert linear frequency to and from mel-frequency), `melbankm` (create a matrix of “filters” for use in computing a mel-scale spectrum from a magnitude DFT spectrum), and `spgrambw` (plot a black-and-white spectrogram). If you find that the toolbox has not been installed, you can download it from <http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html>.

## Appendix B.6 Boolean Variables

Every matlab variable is also a Boolean variable. Boolean functions interpret every nonzero element as TRUE, and every zero-valued element as FALSE.

MATLAB uses `&` for the boolean and operator, `|` for or, and `~` for not. `any(x)` returns 1 if any element of `x` is nonzero; `all(x)` returns 1 if all elements are nonzero. For example:

```
>> x = [1 0 2 4] & [0 0 1 i]
x =
     0     0     1     1
>> x = [1 0 2 4] | [0 0 1 i]
x =
     1     0     1     1
>> y = any(x)
y =
     1
>> y = all(x)
y =
     0
```

You can also compare two vectors element-wise with any of six basic relational operators:

```
< less than
> greater than
== equal to
~= not equal to
<= less than or equal to
>= greater than or equal to

>> x = [1 2 3 4 5] <= [5 4 3 2 1]
```

```
x =
1   1   1   0   0
```

It is possible to use relational operators and logical operators to selectively index elements in a matrix. For example:

```
>> x = [-3 -2 0 2 4]
x =
   -3   -2    0    2    4
>> x(x<0) = - x(x<0)
x =
    3    2    0    2    4
>> x(abs(x)<2 | abs(x) > 3) = 16
x =
    3    2   16    2   16
```

Though this notation can be more confusing than a for loop, MATLAB is written such that this operation executes much, much faster than the equivalent for loop.

## Appendix B.7 Control Structures

The basic control structures in matlab are if:then:elseif:else:end, for:end, and while:end. For example:

```
>> a = -2;
>> if a > 0
>>   x = a^2
>> elseif a == 0,
>>   x = i
>> else
>>   x = - a^2
>> end
x =
    4
>> x = 2;
>> while x < 10
>>   x = x^2
>> end
x =
    4
x =
   16
```

The structure “for x = y” sets the index variable x, in turn, equal to every column of y. For example,

```
>> z = [1 2; 3 4]
z =
    1    2
    3    4
>> for x = z, y=x.^2, end;
y =
    1
    9
y =
    4
   16
```

## Appendix B.8 Graphics

Useful functions (please see help files for great detail):

<code>clf</code>	clear the figure
<code>figure(n)</code>	go to figure window number <code>n</code>
<code>subplot(k,m,n)</code>	go to the <code>k</code> th axis in an $m \times n$ array
<code>plot</code>	plot one vector as a function of the other
<code>axis</code>	get or set the axis limits of your plot
<code>title</code>	add text to the title
<code>xlabel</code>	add text to the x axis
<code>ylabel</code>	add text to the y axis
<code>image</code>	plot a matrix as an image, with current colormap
<code>colormap</code>	show the current colormap for image plots
<code>contour</code>	plot a matrix as a contour plot
<code>surf</code>	plot a matrix as a surface plot
<code>waterfall</code>	plot a matrix as a waterfall plot
<code>print</code>	print current plot to a specified printer or image file

All matlab graphics objects are stored as type-value structures in program memory. It is possible to change the font size, orientation, lighting, micro-positioning, paper position, and a very large number of other properties of a graph by means of its handle. Here are some useful functions:

<code>h=gcf</code>	set <code>h</code> equal to the handle of the current figure window
<code>h=gca</code>	set <code>h</code> equal to the handle of the current plot axes
<code>h=xlabel(...)</code>	create an x label, and set <code>h</code> to its handle
<code>get(h)</code>	return all properties of the object with handle <code>h</code>
<code>get(h, 'Prop')</code>	return <code>h</code> 's value of property <code>Prop</code>
<code>set(h, 'Prop', val)</code>	set <code>h</code> 's value of <code>Prop</code> to <code>val</code>

For example, if you wish to label the ordinate of the current axes with the labels A, B, and C, but you wish these labels to be applied at data values 25, 40, and 45, you could type the following:

```
>> set(gca, 'YTick', [25 40 45]);
>> set(gca, 'YTickLabel', ['A'; 'B'; 'C']);
```

## Appendix B.9 Scripts and Functions

A script is a text file containing a series of matlab commands. By default, scripts and functions end in extension `.m`. If textfile `foo.m` contains some commands, then they can all be run, all at once at the current level of the function stack, by typing

```
>> foo
```

A function file is just like a script file, except that the first line has the following special form:

```
function function-name(argument1, argument2,...)
```

For example, if `foo.m` contains the following text:

```
function z = foo(x,y)
% foo(x,y):
% A function to fooble x and y
z = x .* y
```

Then, in your matlab window, you could type

```
>> A = foo([1 2], [3 4])
A =
    3     8
```

Any line that starts with % is a comment. Any comments that come at the beginning of a script file, or immediately after the `function` line in a function file, are interpreted as the help text for that file. For example

```
>> help foo
foo(x,y):
A function to fooble x and y
```