Lecture 4: Phonetic Features, Neural Nets, and Support Vector Machines

Lecturer: Mark Hasegawa-Johnson (jhasegaw@uiuc.edu)
TA: Sarah Borys (sborys@uiuc.edu)
Web Page: http://www.ifp.uiuc.edu/speech/courses/minicourse/

May 25, 2005

1 Software Tools

In order to do the labs in this section, you will need to install HTK, PVTK, libSVM, and svm-light. If you are interested in training and testing neural networks (instead of SVMs), you may also want to install and compile QuickNet, or you may want to use the matlab neural networks toolkit.

LibSVM [4] and svm_light [6] are distributed in binary form, so you won't need to compile them. If you're serious about SVMs, you will want both, because both have minor limitations (unless I have just missed these features in the documentation). LibSVM but not svm_light is able to train multi-class SVMs. SVM_light but not LibSVM is able to output the SVM discriminant function for each classified vector. Both libSVM and svm_light are built around two main command-line programs: one to train SVMs (called `svm_learn` or `svm-train`), and one to test SVMs (called `svm_classify` or `svm-predict`). LibSVM also includes a useful pre-processing tool called `svm-scale`, as well as a useful directory full of python tools that automatically search for the best SVM hyperparameters, and plot the result. Both LibSVM and svm_light use similar (but not identical) command line syntax, and similar (but not identical) SVM definition file format. Both use the same file format for data: data must be stored in text files, with one vector per row. The first element in the row is the class label (-1 or 1 in svm_light, any digit in libSVM). Remaining columns are in the form dimension:value, where dimension is the feature number, and value is its value; dimensions not specified are assumed to have a value of zero. For example, the following svm_light input data file stores one positive token $[4, 0, 1.2]$ and one negative token $[0, -3, 0.6]$:

```
1 1:4 3:1.2
-1 2:-3 3:0.6
```

In order to use PVTK and HTK: create a directory called "programs," or "src," or "apps," or something like that. Create subdirectories "apps/htk" and "apps/pvtk." If you are on a system that already has `libHTK.a` available somewhere, e.g., in the directory `/workspace/HTK_V3.1/HTKLib/libHTK.linux.a`, then link these directories into htk with commands like

```
ln -s /workspace/HTK_V3.1/HTKLib apps/htk
ln -s /workspace/HTK_V3.1/bin.linux apps/htk
```

If your system does not have HTK installed, then install it: unpack the HTK archive to get directories "apps/htk/HTKLib" and "apps/htk/HTKTools." Create the directory apps/htk/bin.linux or bin.win32, depending on which machine you're using. If you're on a Windows machine with Visual C installed, you may follow the instructions in htk/HTKLib/htk_htklib_nt.mkf and htk/HTKTools/htk_htktools_nt.mkf — or alternatively, you should be able to install Cygwin (be sure to install all of the development tools, including at least gcc and make), and then you could follow the linux instructions in your cygwin window. If you're on a linux machine, edit HTKLib/Makefile and HTKTools/Makefile so that they contain the following lines:

```
CPU = linux
HTKCC = gcc
HTKCF = -ansi
```

Then type `cd HTKLib; make; cd ../HTKTools; make`.

Now unpack the PVTK archive in apps/pvtk. You should be able to type `cd apps/pvtk; make`. If that doesn't work, check the Makefile to make sure that the HLIBS variable is pointing to the true location of the HTK library archive.

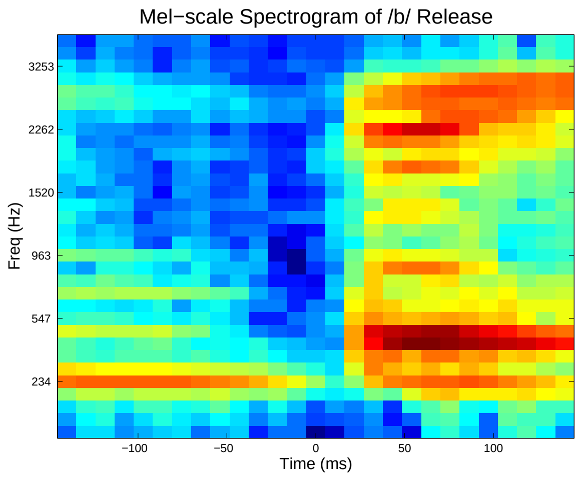
In order to use all of these tools, make sure that these directories are in your path: `export PATH=${PATH}:~/apps/htk/bin`

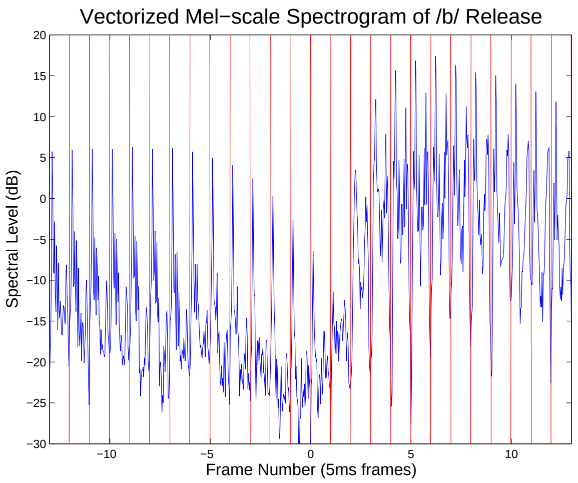
2 Binary Classifiers: LDA, NN, SVM

2.1 Definition of Terms

$$\begin{aligned} \text{Observation:} \quad & \vec{x} \in \Re^K, \quad k^{th} \text{ element is } x_k \\ \text{Label:} \quad & y \in \{-1, 1\} \\ \text{Trainable parameters:} \quad & \vec{\theta} \in \Re^D \\ \text{Training tokens:} \quad & (\vec{x}_m, y_m), \quad \vec{x}_m = [x_{m1}, \ldots, x_{mK}]' \end{aligned}$$

An observation can be a spectral slice, or it can be a whole spectrogram. For example, by concatenating 27 MFSC vectors (from $t-13$ to $t+13$), each with 32 dimensions, we get a total observation vector $\vec{x}_t$ with a dimension of $32 \times 27 = 864$:





A "binary classifier" is a function $h(\vec{x})$ that observes a vector $\vec{x}$ (e.g., a spectrum or spectrogram), and computes a binary output (e.g., the value of a phonological distinctive feature at some specified time):

$$h(\vec{x}|\vec{\theta}) \in \{-1, 1\} \tag{1}$$

Assume that labels and observations obey some constant joint probability distribution, $p(\vec{x}, y)$. The quality of a classifier is it's "expected risk" or "risk" or "expected test corpus error rate:"

$$R(\vec{\theta}) = E|y - h(\vec{x}|\vec{\theta})| = \sum_y \int |y - h(\vec{x}|\vec{\theta})| p(\vec{x}, y) dx \tag{2}$$

In most practical cases, $p(\vec{x}, y)$ is unknown, therefore the expected risk is unknown. Instead, all that we have available are M labeled training tokens, $(y_m, \vec{x}_m)$. Given the training tokens, all that we can compute is the "empirical risk" or "training corpus error:"

$$R_{emp}(\vec{\theta}) = \frac{1}{M} \sum_{m=1}^{M} |y_m - h(\vec{x}_m|\vec{\theta})| \tag{3}$$

2.2 Maximum A Posterior Classification

If the joint probability density $p(\vec{x}, y)$ is known, then the classifier risk can be explicitly minimized. The optimal classifier is the classifier that chooses a label, y, in order to maximize the *a posteriori* probability $p(y|\vec{x})$:

$$E|y - h(\vec{x})| = \int_{h(\vec{x})=-1} p(y = 1|\vec{x})dx + \int_{h(\vec{x})=1} p(y = -1|\vec{x})dx \tag{4}$$

$$h_{opt}(\vec{x}) = \arg\max_{y} p(y|\vec{x}) = \begin{cases} 1 & p(y = 1|\vec{x}) \text{ is larger} \\ -1 & p(y = -1|\vec{x}) \text{ is larger} \end{cases} \tag{5}$$

The MAP classifier can be written as a threshold operation, applied to a real-valued discriminant function $\mathcal{L}(\vec{x})$:

$$h(\vec{x}) = \text{sign}\left(\mathcal{L}(\vec{x}) - b\right) \tag{6}$$

$$\mathcal{L}(\vec{x}) = \log\left(\frac{p(\vec{x}|y = 1)}{p(\vec{x}|y = -1)}\right) \tag{7}$$

$$b = \log\left(\frac{p(y = -1)}{p(y = 1)}\right) \tag{8}$$

2.2.1 MAP Example: Gaussians with Equal Covariance

Suppose we assume that $p(\vec{x}|y)$ is Gaussian, with a covariance matrix R that is independent of the value of y:

$$p(\vec{x}|y) = \frac{1}{\sqrt{2\pi|R|}} e^{-0.5(\vec{x}-\vec{\mu}_y)' R^{-1} (\vec{x}-\vec{\mu}_y)} \tag{9}$$

Then the MAP classifier is a Linear Discriminant:

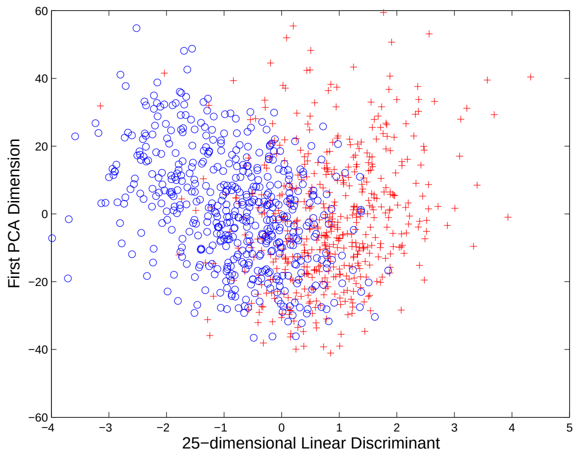
$$h(\vec{x}) = \text{sign}\left(\vec{v}^T \vec{x} - b\right) \tag{10}$$

where the Linear Discriminant vector v and threshold b are:

$$\vec{v} = R^{-1}(\vec{\mu}_1 - \vec{\mu}_{-1}) \tag{11}$$

$$b = \log\left(\frac{p(y = -1)}{p(y = 1)}\right) + 0.5\vec{v}^T(\vec{\mu}_1 - \vec{\mu}_{-1}) \tag{12}$$

For example, consider the problem of classifying stop place of articulation (alveolar vs. non-alveolar) based on 864-dimensional spectrogram observation vectors. The linear discriminant projection of all samples extracted from the TIMIT/TRAIN directory looks like this:

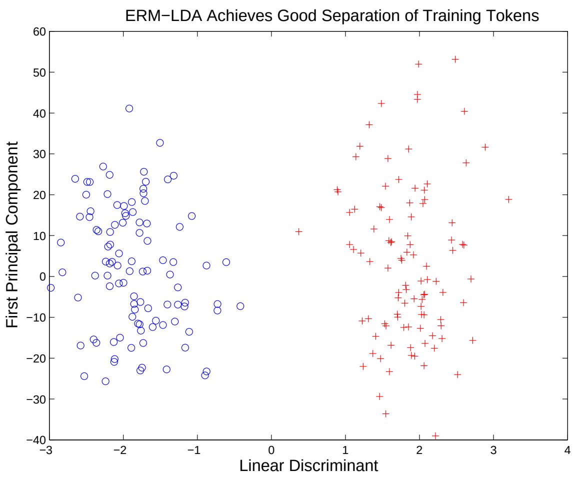


2.2.2 Other Linear Classifiers: Empirical Risk Minimization (ERM)

It is possible to choose a linear discriminant vector, $\vec{v}$, according to any optimality criterion. For example, consider a one-node "neural network;" a one-node neural network computes the function shown in Eq. 10. A neural network is usually trained by choosing some initial value of the coefficient vector $\vec{v}$, and then iteratively adjusting the coefficients in order to reduce classification error on the training corpus. If training is successful, the neural network will learn a vector $\vec{v}$ that minimizes the training-corpus error $R_{emp}(\vec{v}, b)$:

$$\text{Training Rule: Choose } (\vec{v}, b) = \arg\min R_{emp}(\vec{v}, b) \tag{13}$$

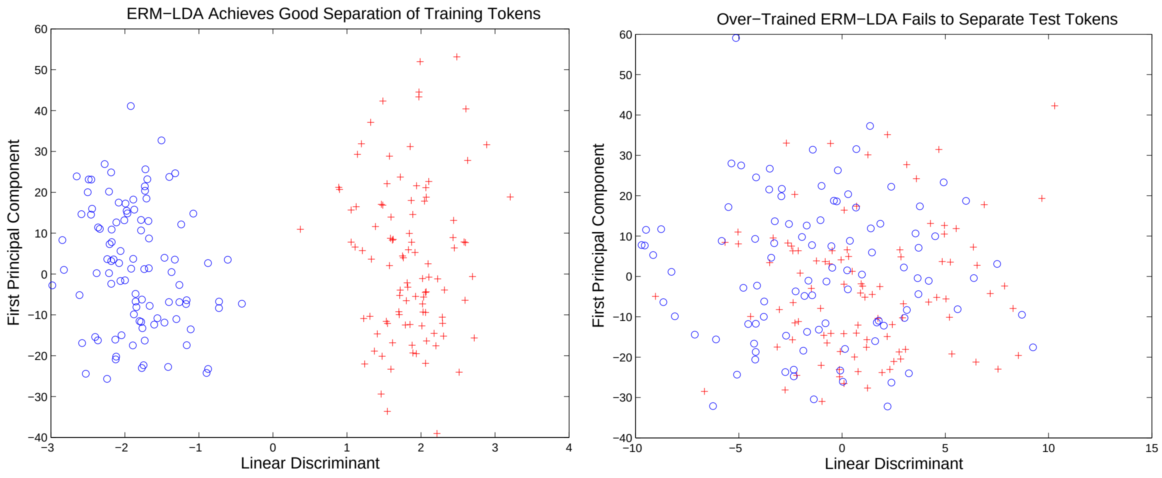
$$\text{...where } R_{emp}(\vec{v}, b) = \frac{1}{M}\sum_{m=1}^{M} |y_m - h(\vec{x}_m|\vec{v}, b)|, \quad \text{...and } h(\vec{x}|\vec{v}, b) = \text{sign}\left(\vec{v}^T\vec{x} - b\right) \tag{14}$$



2.2.3 Over-Training

If the true probability distribution $p(\vec{x}, y)$ is unknown, then, in order to design an MAP classifier, it must be estimated from training tokens. A PDF model learned from training tokens may be incorrect. If the PDF is represented by a family of functions with too many degrees of freedom, the learned PDF will learn accidental details of the distribution of training tokens that are not characteristic of the true underlying PDF. The resulting classification function $h(\vec{x}, y)$ may achieve very low error on the training database, but the low training error may not generalize to novel test tokens drawn from the same underlying probability distribution $p(\vec{x}, y)$.

An ERM classifier (Sec. 2.2.2) suffers similar problems. The vector $\vec{v}$ that is optimal for minimizing training corpus error may not also be optimal for minimizing the expected test corpus error. For example:



2.3 Structural Risk Minimization

2.3.1 Generalization Error and VC Dimension

The objective of machine learning is to minimize the structural risk, not the empirical risk:

$$(\vec{v}, b) = \arg\min R(\vec{v}, b) \tag{15}$$

where

$$R(\vec{v}, b) = E_{p(\vec{x},y)}|y - h(\vec{x}|\vec{v}, b)|, \tag{16}$$

for the unknown true distribution $p(\vec{x}, y)$.

The difference between expected and empirical risk is called the "generalization error:"

$$R(\vec{v}, b) = R_{emp}(\vec{v}, b) + \text{Generalization Error}, \quad R_{emp} = \text{Known}, \ \text{Generalization Error=Unknown} \tag{17}$$

Vapnik and Chervonenkis [12, 2] showed that generalization error is bounded by a nearly-linear function of the VC Dimension, D_{VC}:

$$\mathcal{P}\left\{R(\vec{v}, b) \leq R_{emp}(\vec{v}, b) + f\left(\frac{D_{VC} - \log\delta}{M}\right)\right\} \geq 1 - \delta \tag{18}$$

where $f(\cdot)$ is monotonically increasing and roughly linear. The VC dimension is a measure of the flexibility of the classifier function $h(\vec{x}|\theta)$. If, by varying the parameter vector θ, it is possible to label an arbitrarily large training corpus in a large number of different ways, then the classifier has a high VC dimension. If, on the other hand, the classifier function is limited to only a few different possible "shatterings" of the training data, then the VC dimension is small.

The VC dimension of a linear classifier is strictly less than the number of trainable parameters:

$$\text{Linear Classifier: } D_{VC} \leq K + 1, \quad K = \text{length}(\vec{v}) = \text{length}(\vec{x}) \tag{19}$$

Eq. 19, combined with Eq. 18, suggests the following rule: in order to minimize $R(\vec{\theta})$, we should choose a classifier that has (1) the lowest possible empirical risk, but also (2) the smallest possible parameter dimension. The tradeoff between the parameter dimension of the classifier, and its empirical risk, is captured by the Bayesian Information Criterion (BIC) [9]. By optimizing the BIC, it is possible to choose among classifiers with many different levels of complexity.

Vapnik demonstrated that, in many cases, the upper bound in Eq. 19 is too large; that the true VC dimension of a linear classifier may be much lower. Consider the following situation. For any given training corpus X, normalize v, b so

$$\min_m |\vec{v}^T\vec{x}_m - b| = 1 \tag{20}$$

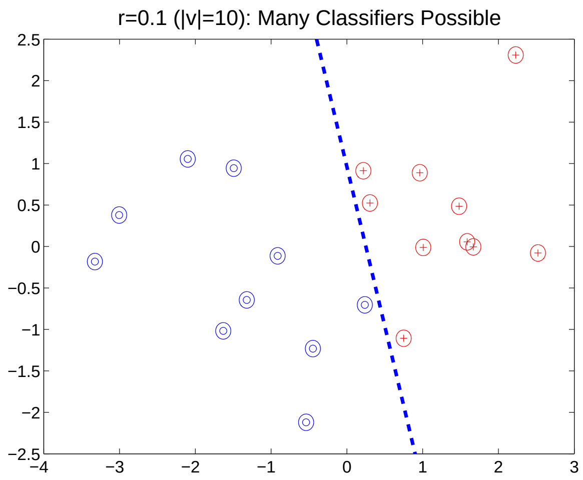
Eq. 20 says that the minimum distance between the hyperplane and any individual data point is $r = 1/|\vec{v}|$. Let R be radius of a ball encircling all of the data points $\vec{x}_m$. Then

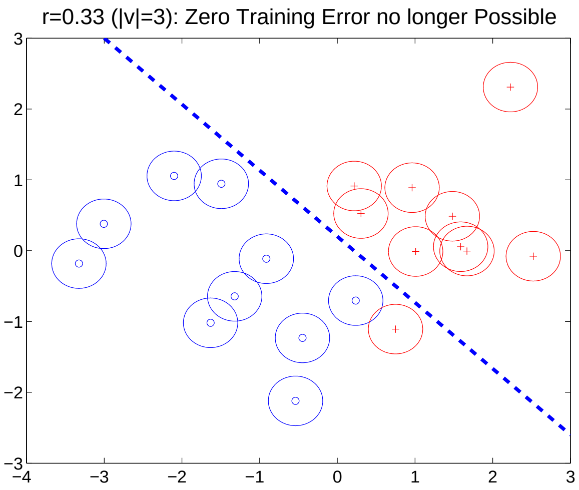
$$D_{VC} \leq \left(\frac{R}{r}\right)^2 = (R|\vec{v}|)^2 \tag{21}$$

According to Eq. 21, it is possible to control the generalization error of the classifier by controlling the magnitude of $\vec{v}$, subject to the constraint in Eq. 21:

$$R(\vec{v}, b) \leq R_{emp}(\vec{v}, b) + R^2|\vec{v}|^2 \tag{22}$$

Schematically, $|\vec{v}|$ controls the expressiveness of the classifier, and a less expressive classifier is less prone to over-training:





2.3.2 Linear Support Vector Machine

Notice that, in any linear classifier, a classification error occurs if $y_m(\vec{v}^T\vec{x}_m - b) < 0$. Define a "partial error" as $y_m(\vec{v}^T\vec{x}_m - b) < 1$, and define the "slack variable" ξ_m as:

$$\xi_m = \max\left(0, 1 - y_m(\vec{v}^T\vec{x}_m - b)\right) \tag{23}$$

Then the training corpus error is bounded by

$$R_{emp}(\vec{v}, b) \le \sum_{m=1}^{M} \xi_m \tag{24}$$

Combining Eq. 24 with Eq. 22, we find that the structural risk of a linear classifier is bounded, with high probability, by

$$R(\vec{v}, b) \le R^2|\vec{v}|^2 + \sum_{m=1}^{M} \xi_m \tag{25}$$

The left-hand-side of Eq. 25 is the optimality criterion minimized by a support vector machine.

The goal of training an SVM, thus, is to minimize Eq. 25, subject to the constraint in Eq. 23. This particular constrained optimization problem is called a "quadratic programming" problem.

$$\text{Minimize:} \qquad (\vec{v}, b) = \arg\min \frac{1}{2}|\vec{v}|^2 + C\sum_{m=1}^{M}\xi_m \tag{26}$$

$$\text{Subject to:} \qquad \xi_m = \max\left(0, 1 - y_m(\vec{v}^T\vec{x}_m - b)\right) \tag{27}$$

Equations 26 and 27 can be transformed into the following equivalent quadratic programming problem: find the coefficients α_m such that

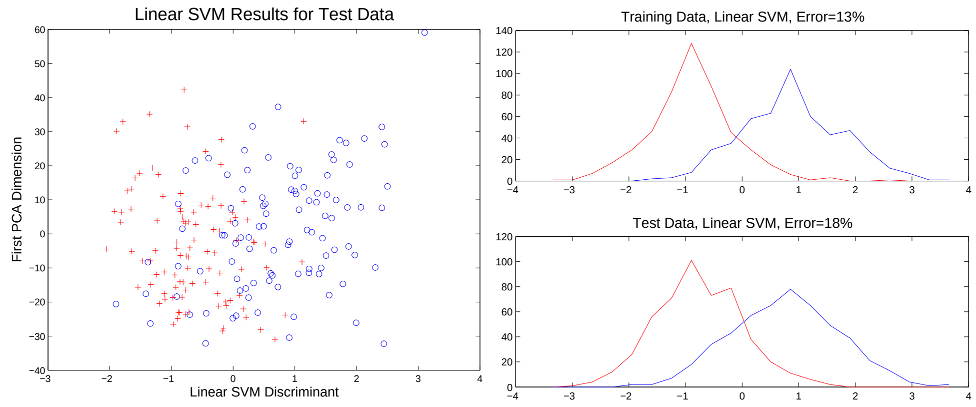
$$\vec{v} = \sum_{m=1}^{M} \alpha_m \vec{x}_m \tag{28}$$

where

$$\alpha_m = \arg\min \sum_m \sum_n \alpha_m \vec{x}'_m \vec{x}_n \alpha_n - \sum_m y_m \alpha_m \tag{29}$$

$$\text{subject to...} \qquad \sum_{m=1}^{M} \alpha_m = 0, \quad 0 \le y_m\alpha_m \le C \tag{30}$$

By minimizing the structural risk (Eq. 25) instead of the empirical risk (Eq. 13), the SVM avoids over-training on data in which either (1) there are too few training tokens, or (2) the observation vector is too large for a standard classifier (LDA or neural network) to work well.



2.3.3 From Linear to Nonlinear SVM

Notice that, according to Eq. 28, the optimum discriminant vector $\vec{v}$ is a weighted combination of the training tokens. Therefore the classifier function $h(\vec{x})$ can be written as

$$h(\vec{x}) = \text{sign}(\vec{v}^T\vec{x} - b) = \text{sign}\left(-b + \sum_{m=1}^{M} \alpha_m \vec{x}_m^T \vec{x}\right) \tag{31}$$

Eq. 31 shows that $h(\vec{x})$ depends only on the dot-products between training vectors $\vec{x}_m$ and the unknown test vector $\vec{x}$. But there is no reason why the dot-product needs to be the only allowable way that we can combine pairs of vectors. In fact, it is possible to use any symmetric, positive-definite function $K(\vec{x}_m, \vec{x})$:

$$h(\vec{x}) = \text{sign}\left(-b + \sum_{m=1}^{M} \alpha_m K(\vec{x}_m, \vec{x})\right) \tag{32}$$

A common, flexible, and extremely useful case is the RBF support vector machine, defined by the kernel function

$$K(\vec{x}_m, \vec{x}) = e^{-\gamma|\vec{x}_m - \vec{x}|^2} \tag{33}$$

The resulting classifier function is

$$h(\vec{x}) = \text{sign}(g(\vec{x}) - b) \tag{34}$$

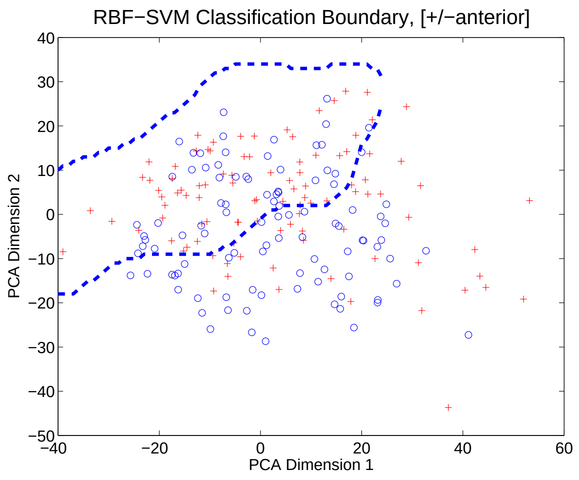
where the nonlinear discriminant function $g(\vec{x})$ is defined as

$$g(\vec{x}) = \sum_{m=1}^{M} \alpha_m K(\vec{x}_m, \vec{x}) \tag{35}$$

A general RBF classifier is infinitely flexible: it is possible for an RBF classifier to "fracture" a training corpus in an infinite number of ways, therefore there is no theoretical upper bound on its VC dimension. An RBF classifier trained using the SVM criterion (Eq. 21), however, has a constrained VC dimension, and therefore it is possible to use the SVM training method to learn an RBF classifier with very low generalization error:

$$D_{VC} \le f(|\vec{v}|^2) = f\left(\sum_m \sum_n \alpha_m \alpha_n K(\vec{x}_m, \vec{x}_n)\right) \tag{36}$$

Here is an example of the classification boundary computed by an RBF classifier:



2.3.4 Practical Training of SVMs

In practice, most people just use svm_light or libSVM to train their SVMs. One needs to choose the "hyper-parameters" γ and C. According to Eq. 21, C should be roughly $1/R$, where R is the radius of a ball containing all of the training tokens. If the training tokens are first normalized using `svm-scale`, then the best value of C should be somewhere around $C \approx 1$. The RBF kernel function is $e^{-\gamma \vec{x}_m^T \vec{x}}$, so γ should be on the order of one over the typical value of $\vec{x}_m^T \vec{x}$; if the data are first normalized to unit standard deviation, or scaled to unit maximum amplitude, then γ should be on the order of $\gamma \sim 1/K$, where K is the dimension of the data vector. In practice, the best values of C and γ may differ from 1 and $1/K$ by one or two orders of magnitude, therefore it is often useful to find the optimum hyper-parameters by experimenting using held-out "development test" data.

3 Phonological Distinctive Features

A phonological feature is a binary-valued label, $y \in \{-1, 1\}$, that distinguishes one set of phonemes from another. Distinctive features were originally based on perceptual characteristics of phonemes [5, 8], then were redefined in terms of the articulatory or speech production characteristics of phonemes [3]; Stevens [10, 11] claims that, in order to be used commonly by the languages of the world, a distinctive feature must have compact articulatory correlates (compact=capable of being distinguished using a low-dimensional classifier) and compact perceptual correlates. Table 1 lists one set of distinctive features that may be useful. "Sonorant" is a sound that may be sung. "Continuant" is a sound with uninterrupted airflow through the vocal tract. "Labial" is made with the lips. "Anterior" is made anterior to the alveolar ridge. "Front" requires a fronted tongue body position; this feature is only distinctive in English for glides (`[+sonorant,+continuant]` phonemese). "Strident" involves high-amplitude turbulence, and "voiced" has vocal fold vibration; these features are only distinctive in English for `[-sonorant]` consonants. Stevens [11] argues that /hh/ is `[+sonorant]`.

"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 [1], and (2) those who believe that every possible acoustic measurement carries information about every possible distinctive feature [7]. 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

phone	sonorant	continuant	labial	anterior	front	strident	voiced
w	+	+	+	+	-		
l	+	+	-	+	-		
r	+	+	-	-	-		
y	+	+	-	-	+		
m	+	-	+	+			
n	+	-	-	+			
ng	+	-	-	-			
b	-	-	+	+		-	+
d	-	-	-	+		-	+
jh	-	-	-	-		+	+
g	-	-	-	-		-	+
p	-	-	+	+		-	-
t	-	-	-	+		-	-
ch	-	-	-	-		+	-
k	-	-	-	-		-	-
f	-	+	+	+		-	-
th	-	+	-	+		-	-
s	-	+	-	+		+	-
sh	-	+	-	-		+	-
hh	-	+	-	-		-	-
v	-	+	+	+		-	+
dh	-	+	-	+		-	+
z	-	+	-	+		+	+
zh	-	+	-	-		+	+

Table 1: An example distinctive feature notation for the consonants of English.

"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, $250 \leq F_1 \leq 1000$Hz, $900 \leq F_2 \leq 2400$Hz, $2200 \leq F_3 \leq 3000$Hz; F3 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.

4 Laboratory Exercise

Install HTK, PVTK, and libSVM on your own computer. From the course web page, download the ToBI-labeled Switchboard files [13]. Within the top-level directory (ToBIlabeling), you should find four subdirectories: sw23ToBIlabeling (containing the waveforms and TextGrid files), scp (containing HTK script files), mlf (containing HTK master label files), and cfg (containing an HTK configuration file, suitable for use with HCopy).

The script files "HCopy1.scp" and "HCopy2.scp" each contain pairs of files: an input waveform, and an output PLP file. HCopy1.scp lists the first 45 files in the database (a kind of "training" dataset), while HCopy2.scp lists the remaining 44 files.

Create the subdirectory "plp" that will hold the converted PLP files. Run `HCopy` with no arguments, in order to read information about `HCopy`. Now run `HCopy` with script-file and config-file arguments, in order to generate PLP output files from your waveform inputs. Use `HList` to browse one or two of the PLP output

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	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
	fricative	frication spectrum (during closure) has very low amplitude, with roughly equal energy at all frequencies above 1000Hz
	stop release	burst spectrum (at t=0ms) has very low amplitude, with roughly equal energy at all frequencies above 1000Hz
	stop release	VOT is shorter than predicted by voicing features
blade	any	formant loci are $1600 < F_2 < 2000$Hz, $F_3 \approx 3000$Hz
	stop release	burst spectrum is low amplitude and highpass, with more energy at high frequencies (above 2500Hz) than at low frequencies (below 2000Hz)
	fricative	frication spectrum is low-amplitude and highpass
body	any	formant loci may be at any frequency at all, but F2 locus and F3 locus are always very close together
	stop release	burst spectrum shows a compact peak (on the spectrogram, a lump of visible energy) between 1000Hz and 3500Hz
	fricative	frication spectrum shows a compact peak
anterior	any	`[-anterior]` phones have F2 and F3 formant loci close together, `[+anterior]` phones have F2 and F3 loci far apart
	stop or fricative	`[-anterior]` phones have a compact frication or burst peak between 1000Hz and 3500Hz, `[+anterior]` 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	`[+voiced]` consonants are shorter than `[-voiced]` consonants
	any closure	all stops and fricatives (both voiced *and unvoiced*) 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
	fricative release	voiced fricatives, but *not stops*, may have a voice bar for a few frames before release
	stop release	unvoiced stops have a VOT longer than 25ms

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

files.

The script files "VExtract1.scp" and "VExtract2.scp" are exactly equal to the second columns of "HCopy1.scp" and "HCopy2.scp" — that is, they only list the PLP filenames. Run `VExtract` with no arguments, in order to read information about `VExtract`. Now run `VExtract` with arguments, in order to extract tokens corresponding to particular phonemes. For example, if you wanted to just compare /r/ and /l/ with /s/ and /sh/, you could type

```
VExtract -T 1 -b -p /r/ /+1/ -p /l/ /+1/ -p /s/ /-1/ -p /sh/ /-1/ \
  -o group1.toks -I mlf/phn_wrd_tones.mlf -S scp/VExtract1.scp \
  &> group1.log;

VExtract -T 1 -b -p /r/ /+1/ -p /l/ /+1/ -p /s/ /-1/ -p /sh/ /-1/ \
  -o group2.toks -I mlf/phn_wrd_tones.mlf -S scp/VExtract2.scp \
  &> group2.log;
```

The command line options have the following meanings.

- The "-T" option tells VExtract to output trace (debugging) information. Specifically, it will output the frame number and filename of every vector that it selects from the database, along with the label it believes appropriate. The notation `&> group1.log` at the end will cause all of the debugging information to be saved to the file group1.log (if your unix system is using bash to parse the command line. If the command above doesn't work, try typing "bash" first to get a bash parser, then re-enter the line above).
- Each of the "-p" options specifies a phoneme label that VExtract should look for, and the class (+1 or -1) to which that phoneme belongs. The "-b" option specifies that these phoneme labels should be surrounded by word boundaries (whitespace or newline), e.g., so that the symbol "er" will not count as equivalent to "r".
- The "-o" option specifies the SVM tokens file to which data should be stored.
- The "-S" option specifies a script file that lists the PLP files from which data should be extracted.
- The "-I" option specifies that VExtract should read through the file `mlf/phn_wrd_tones.mlf` in order to find example phonemes. Open this file with a text editor, and take a look. This file contains transcriptions of all of the ICSI Switchboard utterances, not just the prosodically transcribed ones. If you want to see how the prosodically transcribed ones come out, search through the file for the symbol %. This file was created by using `rnc2mlf.pl` to convert the ICSI Switchboard phn transcriptions, `praat2mlf.pl` to convert the Illinois Switchboard TOBI transcriptions, and `layermlfs.pl` to combine the two.

Read through the log files, `group1.log` and `group2.log`. Open the MLF, `phn_wrd_tones.mlf`. Check a few of the extracted vector times, in order to make sure that they really do correspond to the phoneme labels that you were trying to extract. Use `HList` to find one such frame in the original PLP file, and compare the numbers in the PLP file with the numbers in `group1.toks`, in order to make sure that they are the same.

VExtract is not yet bug-free. Most likely, if you have followed instructions so far, your files group1.toks and group2.toks have nans in them (not-a-number entries). Open one with a text editor, and take a look. NaNs will screw up libSVM or svm_light, so you need to get rid of them. Try the following, to set all nans to zero (and thereby get them out of the way):

```
cp group1.toks foo.toks; sed 's/nan/0/g' foo.toks > group1.toks;
cp group2.toks foo.toks; sed 's/nan/0/g' foo.toks > group2.toks;
```

Now, use libSVM to scale your tokens. First, decide which toks file will be training data, and which will be testing data. Suppose that you decide to use group2.toks as training data, and group1.toks as testing data. In order to compile the scaling statistics, and scale group2, enter

```
svm-scale -l -1 -u 1 -s scaling.stats group2.toks > g2_scaled.toks;
```

Now, you can use the same scaling statistics (compiled from the training tokens) in order to scale the testing tokens (because if you're going to scale the data, you'd better scale both training and testing data equally!!). Type

```
svm-scale -r scaling.stats group1.toks > g1_scaled.toks;
```

... in order to read in scaling stats from `scaling.stats`, and apply them to group1.toks.

Train a linear SVM, using the default value of C. Type

```
svm-learn -t 0 g2_scaled.toks my.svm
```

Open the file `my.svm` with a text editor. The header specifies the kernel type (linear or RBF), the number of classes, and the number of support vectors (the number of training vectors that are used to compute the discriminant function). After the codeword `SV`, the support vectors themselves are stored.

Test the SVM on the training data, then on the independent test data:

```
svm-predict g2_scaled.toks my.svm g2.out;
svm-predict g1_scaled.toks my.svm g1.out;
```

How well did you do?

Warning: I have found that, because of the very small size of this training corpus, I sometimes get quite poor phoneme classification performance. Please don't be discouraged if that happens to you...

References

[1] Sheila E. Blumstein and Kenneth N. Stevens. Acoustic invariance in speech production: Evidence from measurements of the spectral characteristics of stop consonants. *Journal of the Acoustical Society of America*, 66(4):1001–1017, October 1979.

[2] C.J.C. Burges. A tutorial on support vector machines for pattern recognition. *Knowledge Discovery and Data Mining*, 2(2), 1998.

[3] N. Chomsky and M. Halle. *The Sound Pattern of English*. Harper and Row, New York, NY, 1968.

[4] R.-E. Fan, P.-H. Chen, and C.-J. Lin. Working set selection using the second order information for training svm. Technical report, Department of Computer Science, National Taiwan University, 2005.

[5] R. Jakobson, G. Fant, and M. Halle. Preliminaries to speech analysis. Technical Report 13, MIT Acoustics Laboratory, 1952.

[6] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In *Proc. of European Conference on Machine Learning*, 1998.

[7] Diane Kewley-Port. Time-varying features as correlates of place of articulation in stop consonants. *Journal of the Acoustical Society of America*, 73(1):322–335, Jan. 1983.

[8] G. A. Miller and P. E. Nicely. Analysis of perceptual confusions among some English consonants. *Journal of the Acoustical Society of America*, 27:338–352, 1955.

[9] G. Schwartz. Estimating the dimension of a model. *The Annals of Statistics*, 5(2):461–464, 1978.

[10] K. N. Stevens. On the quantal nature of speech. *Journal of Phonetics*, 17:3–45, 1989.

[11] Kenneth N. Stevens. *Acoustic Phonetics*. MIT Press, Cambridge, MA, 1999.

[12] Vladimir Vapnik. *Statistical Learning Theory*. Wiley, New York, 1998.

[13] Taejin Yoon, Sandra Chavarria, Jennifer Cole, and Mark Hasegawa-Johnson. Intertranscriber reliability of prosodic labeling on telephone conversation using tobi. In *Proc. Internat. Conf. Spoken Language Processing*, 2004.