From Webster’s Revised Unabridged Dictionary (1913) [web1913]:

Gerrymander \Ger’ry*man”der\, v. t. [imp. & p. p. \{Gerrymandered\}; p. pr. & vb. n. \{Gerrymandering\}.

To divide (a State) into districts for the choice of representatives, in an unnatural and unfair way, with a view to give a political party an advantage over its opponent. [Political Cant, U. S.]

Note: This was done in Massachusetts at a time when Elbridge Gerry was governor, and was attributed to his influence, hence the name; though it is now known that he was opposed to the measure.

-Bartlett.

Gerrymandering:
A Pattern Classifier Based Approach to Campaign Planning

Corey Halpin

December 21, 2001

Contents

1 Introduction 2
   1.1 Problem Statement ................. 2
   1.2 Motivation .......................... 2
   1.3 Baseline ............................ 2

2 Work Performed 2
   2.1 Data Acquisition .................. 2
      2.1.1 Census Data .................. 2
      2.1.2 Vote Results ................ 3
      2.1.3 The Search for More ........ 3
   2.2 Device Design .................... 3
      2.2.1 MLP .......................... 3
      2.2.2 GMM .......................... 3
      2.2.3 KNN .......................... 4
      2.2.4 MOE .......................... 4

3 Results 4

4 Discussion 4

5 Code 4
   5.1 MLP ............................... 4
      5.1.1 bptrain.m .................... 4
      5.1.2 bptest.m ..................... 5
      5.1.3 actfun.m ........................ 5
   5.2 GMM ............................... 6
      5.2.1 mltrainnew.m ............... 6
      5.2.2 mltestnew.m ................ 7
      5.2.3 mlgnm.m ..................... 7
      5.2.4 gnm.m ........................ 7
      5.2.5 gnmactiv.m .................. 7
      5.2.6 gnmem.m ..................... 8
      5.2.7 gnmprob.m ................... 9
      5.2.8 kmeansf.m ................... 9
      5.2.9 kmeantest.m ................. 10
      5.2.10 kmeansprob.m ............. 10
      5.2.11 dist.m ...................... 10
      5.2.12 consist.m ................... 10
   5.3 KNN ............................... 11
      5.3.1 knn.m .......................... 11
   5.4 MOE ............................... 11
      5.4.1 moe.m .......................... 11
      5.4.2 loadstate.m ................. 12
      5.4.3 savestate.m .................. 13
      5.4.4 scale.m ...................... 13
      5.4.5 simplify.m ................... 13
      5.4.6 remove.m ..................... 14
      5.4.7 split.m ...................... 14
   5.5 Matlab Compatibility ............. 14
      5.5.1 nth.m .......................... 14
      5.5.2 list.m .......................... 14
1 Introduction

1.1 Problem Statement

American Presidential campaigns now have increasingly large budgets. I was unable to find exact numbers, but according to some sources Bush spent at least $100 million dollars on the last campaign.

The spending of this money is largely directed by “Campaign Consultants”, whose expertise ranges from being past political contenders, to interns just out of college. From what I have found they employ a number of heuristic solutions and a lot of blind, dumb luck to divine how the population of a given region will vote when presented with certain choices. It would be nice to have a more rigorous method, if not a more reliable one.

In my research of the matter, the only mathematical predictor of general elections is the “Presidential Vote Equation”. This provides an estimate of the over all popular vote, which is not what always elects a president. It fails to take into account the strange Electoral College system inherent in American Politics. Also, this system is not capable of finer granularity than the entire nation popular vote. If you ask it for an estimate of the results for a single state, it gives you a dumb look.

It would be nice to have an inexpensive, fairly accurate, easy to use, artificial intelligence based device to do some of this work. The goal of this is not to replace human campaign planners, but to give them a tool that they can use to check their estimations.

1.2 Motivation

The success or failure of a campaign depends on how it directs its advertising. Potentially tens of millions of dollars could be saved by better direction of advertising funds, and by better planning of campaign stops. More appropriate use of campaign time could greatly increase the effectiveness of the candidates stops.

1.3 Baseline

The best, and only vote predictor that I found was the Presidential Vote Equation, the source of which is Reference 6 in the bibliography.

The equation takes the following form:

\[ V = a_1 + a_2 \cdot I + a_3 \cdot I \cdot D_{\text{war}} + a_4 \cdot g_3 \cdot I + a_5 \cdot p_{15} \cdot I \cdot (1 - D_{\text{war}}) + a_6 \cdot n \cdot I \cdot (1 - D_{\text{war}}) + a_7 \cdot D_{\text{per}} + a_8 \cdot D_{\text{ur}} \]

Where:

- \( D_{\text{ur}} \) = 0 if the incumbent party has been in power for one term. 1 for Democrats and -1 for Republicans if the incumbent party has been in power for two consecutive terms, 1.25 for Democrats and -1.25 for Republicans if the incumbent party has been in power for three consecutive terms, 1.50 and -1.50 for four consecutive terms, and so on.
- \( D_{\text{war}} \) = 1 for the elections of 1920, 1944, and 1948, and 0 otherwise.
- \( g_3 \) = growth rate of real per capita GDP in the first three quarters of the election year (annual rate).
- \( p_{15} \) = absolute value of the growth rate of the GDP deflator in the first 15 quarters of the administration (annual rate).
- \( n \) = number of quarters in the first 15 quarters of the administration in which the growth rate of real per capita GDP is greater than 3.2 percent at an annual rate.

The constants a are empirically determined. A table can be found on the website cited above.

This equation predicted that in 1996, the Democrats would receive 54.74% of the vote. My model predicts 62.74% would go to the Democrats. Mine was correct to four digits. The PVE prediction was off by 8%. Also, the PVE does not allow for state by state, or region by region prediction in the manner that my model does.

2 Work Performed

2.1 Data Acquisition

2.1.1 Census Data

Abstracts of the decennial census can be obtained from http://www.census.gov. These are not terribly useful, however. The 2000 Census has not been fully released. The 1990 Census has been released, but one needs to wade through the html forms interface to extract what tables one wants. After some work, it is possible to extract meaningful information from the 1990 Census data.

There are population estimates done every two years, but the data listed on the website is incomplete for these estimates at best.

There are economic censi done every five years, but they offer only information about what businesses are succeeding in what regions and by how much.

The population data that I used as input vectors for my classifier was based on census1990, and the overall population change from 1990 to 2000 per state. I assumed that each state’s population scaled in the same proportions from 1990 to 92 and to 96. This was probably a bad assumption, but I could find no other way.
2.1.2 Vote Results

The national records archive maintains an excellent database of the results of past elections. The electoral and popular vote information that I required could be found there.

2.1.3 The Search for More

It would be nice to take into account what the campaign manager does, not just the population demographics. Unfortunately, I was unable to find campaign budget information for all the candidates in either of the elections that I used the classifier on.

I found excellent resources to determine who in power is getting money from where, but could find little or no information about the money that people used to get into power.

I would also like to take into account how many campaign stops the candidate makes in each state. I was able to find this for the 2000 election, but all of my other sources for 2000 were so sparse that I could not use it.

I would like to consider voting age population and voter registration. Unfortunately those numbers are available only from 1996 forward. The closest election to my center of population data is 1992. I consider it a stretch to test with only approximated information for 1996. I did not want to train with data from 1990 on the 1996 results, and then test on the 2000 results. The data wouldn’t have any relevance to the problem that it was being applied to.

2.2 Device Design

I chose to implement my device as a mixture of experts, written in the GNU Octave programming language. Octave is a mostly Matlab compatible language, developed in the free and open source method.

I chose octave because it can run on an ordinary PC. This has many advantages in terms of the cost of this device. Workstations are expensive, licenses to use the operating systems that come on a work station are expensive, Matlab itself is expensive. If I produce a device that can be 90% as accurate as a human campaign consultant at predicting vote outcomes, but requires the annual salary of five such experts solely for it’s installation, then I have constructed an interesting toy, not a usable tool.

Octave is also nice in that the syntax and language it uses is 90% identical to Matlab. Almost all Matlab code can run without modification under Octave. Unfortunately, the code for this course uses many of the advanced features of Matlab which can not be found in Octave. However, all data structures implemented by Octave are implemented in some way by Matlab. This means that any octave code can run under Matlab, with minor modifications.

Initial development was performed on my PC, using octave, and validating results on the iris data set and the wine data set. After this was completed, a Matlab compatibility layer was constructed, and models of the much more complicated Presidential campaign data sets were constructed and tested on CAE UNIX machines under Matlab.

The device has the ability to save state, and reload state later for testing with potentially different data sets. Currently Matlab cannot read Octave generated save state files. Octave can read Matlab generated save state files. If desired, Octave can export it’s save state files to Matlab format.

2.2.1 MLP

I based my implementation of a multi-layer perceptron on the bp.m code and associated fragments written by Prof. Hu.

Octave does not support cell arrays in the way that Matlab does, but does have support for a similar “list” structure. I converted the bp code to use lists. Because I planned to use this in a mixture of experts setting, I re-wrote bp.m and bptest.m into function call versions of the same. Stipulating that there would not be an AI expert on hand when this device is used, I wrote in certain auto-configuration heuristics.

The learning rate parameters were hardcoded to $\alpha = 0.01$ and $\mu = 0.5$. Outputs are scaled from 0.2 to 0.8. Hidden layers are twice the input dimension in width. There are three hidden layers. I know that two is sufficient to solve any problem, but my results were more stabile for all of the testing sets that I used with three. The number of epochs performed is one thousand times the input dimension, and the maximum allowed stall is 100 times the input dimension.

I tried adaptive adjustment of learning rate parameters. This had the effect of never allowing the classifier to achieve 100% Crate. I abandoned the search for algorithms to adjust learning rate after finding the correct scaling of data.

I tried increasing the random perturbation in proportion to the current length of stall. This caused wild and unpleasant oscillations in the Crate, sometimes even not classifying anything as belonging to a class. I abandoned this effort.

I also tried increasing and decreasing the number of hidden layers. The best results were achieved with my current three hidden layers.

Success with this algorithm was achieved after appropriate scaling of data, and selection of the number of hidden layers.

2.2.2 GMM

Again my Gaussian Mixture Model is based on the nlgmm.m code used in class.

Octave does not support tridimensional arrays. To compensate for this, I re-wrote the sections of the gmm code dealing with Covariance matrices such that they would store the matrices in vector-reshaped format into a 2d array and then extract them before performing any operation on them. I had to make similar changes to the list/cell array handling that I implemented for the MLP.

Because I want this to be a largely non-interactive classifier, I needed some heuristic to determine how many clusters
exist. Based on the comments in mlgmm.m, I wrote a fragment of code that will determine how many eigenvalues of approximately the same order exist for the input data, and then use that number of clusters.

I found in the gmmem.m code that the only termination criterion was the number of iterations. Finding this to take an unacceptable amount of time for no improvement whatsoever, I implemented a maxstall counter similar to the one existing in bp.m

I also modified mltestnew.m to return the Cmat and Crate.

2.2.3 KNN

The KNN algorithm actually worked with no modifications.

I am using KNN of order three.

2.2.4 MOE

The expert mixing is a simple weighted average based on the training Crate.

The MOE program itself handles all of the user interaction, file IO and such.

It has the capability to load previous state files, generate test files on the fly (using 20% of the input data), simplify data to remove vectors with low mean, low standard deviation, equivalent vectors, and 2-way linearly dependant vectors (of the form \( v_1 = v_2 + v_3 \)). It scales data appropriately, using log scale if the data has a large range. It remembers what vectors it removed, and by how much it scaled that data so that the state can be loaded and applied to any number of test files.

3 Results

From the Census website, I obtained percentage growth in population per state from 1990 to 1999. Using this, and the census data from 1990, I constructed estimates of the state populations in 1992 and in 1996. I assumed in this construction that the relative numbers of each demographic remained constant.

From the NARA website, I obtained voting results from 1992 and from 1996.

I used these two data sets to construct training (election 1992) and testing (election 1996) data.

<table>
<thead>
<tr>
<th>Training</th>
<th>BP</th>
<th>GMM</th>
<th>KNN</th>
<th>MoE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crate</td>
<td>100%</td>
<td>100%</td>
<td>88.24%</td>
<td>100%</td>
</tr>
<tr>
<td>Cmat</td>
<td>32 0</td>
<td>32 0</td>
<td>30 4</td>
<td>32 0</td>
</tr>
<tr>
<td></td>
<td>0 19</td>
<td>0 19</td>
<td>2 15</td>
<td>0 19</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Testing</th>
<th>BP</th>
<th>GMM</th>
<th>KNN</th>
<th>MoE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crate</td>
<td>100%</td>
<td>100%</td>
<td>86.27%</td>
<td>100%</td>
</tr>
<tr>
<td>Cmat</td>
<td>32 0</td>
<td>32 0</td>
<td>29 4</td>
<td>32 0</td>
</tr>
<tr>
<td></td>
<td>0 19</td>
<td>0 19</td>
<td>3 15</td>
<td>0 19</td>
</tr>
</tbody>
</table>

4 Discussion

I would have liked to have a larger base of training data than I did, however the sparseness of population demographics and campaign fund information made this impossible within the given time frame.

I doubt the accuracy of the population numbers I used. As I said above, they were based on the 1990 census data, times the per-state percent of change in population. This assumes that the relative populations, incomes and such change only once every ten years, which is an obvious erroneous assumption. I could not, however, find any better method to generate this data.

My intent at the outset was to make some kind of ready-for-market tool that could (with appropriate copyright permissions) be packaged the day after I hand in the project. I have not achieved this. However, my success predicting the 1996 election based on population, and the results of the 1992 election leads me to believe that such a tool is feasible, and that with more complete data sets, the tool I have created would be capable of further prediction.

I also note that nowhere in the definition of moe.m does it actually depend on what kind of data is being classified. It generates training and testing sets on the fly from input files. This feature of the design has two primary purposes. The first of these was to facilitate testing on a variety of available data. The census data and associated vote data is a huge data set, and finding errors in the running of it takes some time. Therefore, it is nice to have the ability to select other data sets for validation purposes. A run of simplify.m on the election 92 training data takes two hours on a CAE workstation under matlab, and approximately 18 hours under Octave with a pentium II/300 with 320M of RAM. The second reason for this method of design is that the end user can adapt this device to work on any set of election data, or any other type of data for that matter. It is conceivable that this tool could be used to predict elections based on state census data having a completely different range and number of values than the nation wide census.

5 Code

5.1 MLP

5.1.1 bptrain.m

```matlab
function [Cmat, Crate, wbest, atype] = bptrain(train0, Dim)
[Kr, MN] = size(train0);
N = MN - Dim;
L = 5;
n = zeros(1, L);
n(1) = Dim; n(L) = N;
w = list(pi); dw = list(pi);
for i = 2:L - 1
  n(i) = 2 * Dim;
```
% for
w{L}=0.005*randn(n(L),n(L-1)+1);
dw{L}=zeros(size(nth(w,L)));
end % for

atype=2*ones(L,1); atype(L)=1;
if atype(L)==1, outlow = 0.2;
elseif atype(L)==2 | atype(L) == 3, outlow = -0.8;
end
outhigh=0.8;

[train0(:,Dim+1:MN),zmin,zmax]=scale(...
train0(:,Dim+1:MN),outlow,outhigh);
alpha=0.01;mom=0.5; Crate=0;
nepoch=1000*Dim;
K=min(64,Kr);
nck=ceil(Kr/K);
maxstall=100*Dim;
nstall=0; bstrate=0;
t= 1; ptr=1; not_converged = 1;
while not_converged==1,
  [train,ptr,train0]=rsample(train0,K,Kr,ptr);
  z{1}=(train(:,1:Dim))';
  d=train(:,Dim+1:MN)';
  for l=2:L,
    u{l}=nth(w,l)*[ones(1,K);nth(z,l-1)];
    z{l}=actfun(nth(u,l),atype(l)); % z{l} is n(l) by K
    % for
    error=d-nth(z,L);
    delta(L)=actfunp(nth(u,L),atype(L)).*error;
    if L>2,
      for l=L-1:-1:2,
        tmp=nth(w,l+1);
        delta(l)=(tmp(:,2:n(l)+1))'*nth(delta,l+1)... 
        .*actfump(nth(u,l),atype(l));
      % if L
    end % for
    for l=2:L,
      dw{l}=alpha*nth(delta,l)*[ones(1,K);nth(z(l-1));nth(z(l-1))]'... 
      .*actfump(nth(u,l),atype(l));
    % if
    end % for
    if t>=nepoch;
      not_converged=0;
      disp('-----');
      t, Cmat, Crate
      if Crate > bstrate,
        bstrate=Crate;
        wbest=w;
        nstall=0;
      else
        nstall=nstall+1;
      end % if crate
      if nstall>maxstall,
        not_converged=0;
        disp(['Terminate: no improvement for too long']);
      end % if nstall
      if Crate == 100,
        not_converged=0;
      end % if crate
      if Crate == 100,
        not_converged=0;
        disp('Terminate: classification rate is 100%')
      end
    end % if nck
    t = t + 1;
  end % while loop
  [Cmat,Crate]=bptest(wbest,train0,atype);
end % if not_converged==1

5.1.2 bptest.m

function [Cmat,Crate,class]=bptest(w,test,atype)
% Usage: [Cmat,Crate]=bptest(w,test,atype)
z=list(pi);
if nargin<3, atype=ones(length(w),1); end
[K,MN]=size(test);
M=M1-1; N=MN-M;
L=length(w);
for l=1:L,
  if N==1,
    z{L}=[nth(z,L)>0.5];[nth(z,L)<0.5];
    target=[target;ones(1,K)-target];
    N=2;
  end
  class=[(nth(z,L) - ones(N,1)*max(nth(z,L))) == 0]';
  Cmat=[(target-ones(N,1)*max(target)) == 0]*class;
  Crate=sum(diag(Cmat))*100/K;
end % for

5.1.3 actfump.m

function y=actfun(x,type,par)
% Usage: y=actfun(x,type,par)
if nargin<=2,
  par=1;
end
if nargin==1,
type=1;
end
switch type
  case 1 % sigmoid
    T=par(1);
    y = 1./(1+exp(-x/T));
  case 2 % tanh
    T=par(1);
    tmp=exp(x/T);
    y=(tmp-1./tmp)./(tmp+1./tmp);
  case 3 % linear
    a=par(1); b=par(2);
    y=a*x+b;
  case 4 % radial
    m=par(1); sig=par(2);
    s=sig^2;
    tmp=(x-m).*(x-m);
    y=exp(-tmp/s);
end

5.1.4 actfunp.m

function yp=actfunp(x,type,par)
  switch type
    case 1 % sigmoid
      T=par(1);
      y = 1./(1+exp(-x/T));
      yp = y.*(ones(size(y))-y)/T;
    case 2 % tanh
      T=par(1);
      tmp=exp(x/T);
      y=(tmp-1./tmp)./(tmp+1./tmp);
      yp=(ones(size(y))-y.*y)/T;
    case 3 % linear
      a=par(1); b=par(2);
      y=a*x+b;
    case 4 % radial
      m=par(1); sig=par(2);
      s=sig^2;
      tmp=(x-m).*(x-m);
      y=exp(-tmp/s);
      yp=(-2/sig^2)*(x-m).*y;
  end

5.1.5 rsample.m

function [train,ptr,x]=rsample(x,K,Kr,ptr)
  switch type
    case 1 % sigmoid
      T=par(1);
      y = 1./(1+exp(-x/T));
      train=x(1:K,:);
      ptr=K;
    else
      train=x(ptr+1:ptr+K,:);
      ptr=ptr+K;
  end

5.1.6 randomize.m

function B=randomize(A)
  W= list(pi); Covar= list(pi); cinv= list(pi);
  Cen=[]; clabel=[]; ppri=[];
  for i=1:S,
    disp(['----- Analyzing class ' int2str(i) ... 
          ' of ' int2str(S) ' -----']);
    [Wtmp,Covartmp,priors]=mlgmm(Pr(find(Tr(:,i)==1),:));
    W{i}=Wtmp; Covar{i}=Covartmp;
    if (isempty(Cen)), Cen=nth(W,i);
    else, Cen=[Cen; nth(W,i)]; end
    nc(i)=size(nth(W,i),1);
    for j=1:nc(i),
    [C,M2]=size(tmp);
    M=sqrt(M2);
    tmp2=reshape(tmp(j,:),M,M);
    cinv{length(clabel)+j}=pinv(tmp2);
    end
  if (isempty(clabel)), clabel=i*ones(nc(i),1);
  else, clabel=[clabel; i*ones(nc(i),1)]; end
  if (isempty(ppri)), ppri= diag(diag(priors));
  else, ppri=[ppri; diag(diag(priors))]; end

5.2 GMM

5.2.1 mltrainnew.m

function [Cen,cinv,clabel,nc,ppri]=mltrainnew(Pr,Tr)
  [K,N]=size(Pr);
  [K,S]=size(Tr);
  nos=sum(Tr);
  if S==1,
    Tr=[Tr ones(K,1)-Tr];
  S=2;
  end
  W= list(pi); Covar= list(pi); cinv= list(pi);
  Cen=[]; clabel=[]; ppri=[];
  for i=1:S,
    disp(['----- Analyzing class ' int2str(i) ... 
          ' of ' int2str(S) ' -----']);
    [Wtmp,Covartmp,priors]=mlgmm(Pr(find(Tr(:,i)==1),:));
    W{i}=Wtmp; Covar{i}=Covartmp;
    if (isempty(Cen)), Cen=nth(W,i);
    else, Cen=[Cen; nth(W,i)]; end
    nc(i)=size(nth(W,i),1);
    for j=1:nc(i),
      [C,M2]=size(tmp);
      M=sqrt(M2);
      tmp2=reshape(tmp(j,:),M,M);
      cinv{length(clabel)+j}=pinv(tmp2);
    end
    if (isempty(clabel)), clabel=i*ones(nc(i),1);
    else, clabel=[clabel; i*ones(nc(i),1)]; end
    if (isempty(ppri)), ppri= diag(diag(priors));
    else, ppri=[ppri; diag(diag(priors))]; end

  % Usage: [train,ptr]=rsample(x,K,Kr,ptr)
  if ptr*K > Kr,
    x= randomize(x);
    train=x(1:K,:);
    ptr=K;
  else
    train=x(ptr+1:ptr+K,:);
    ptr=ptr+K;
  end

  % Usage: [train,ptr]=rsample(x,K,Kr,ptr)
  % Usage: [train,ptr]=rsample(x,K,Kr,ptr)
  % Usage: [train,ptr]=rsample(x,K,Kr,ptr)
  % Usage: [train,ptr]=rsample(x,K,Kr,ptr)
  % Usage: [train,ptr]=rsample(x,K,Kr,ptr)
5.2.2 mltestnew.m

function [Cmat,Cr,cl,conf]=...
mltestnew(Pt,Tt,Cen,cinv,clabel,nc,ppri)
% Usage: [Cmat,Cr,cl,conf]=
% mltestnew(Pt,Tt,Cen,cinv,clabel,nc,ppri)

[Q,N]=size(Pt);
[Q,S]=size(Tt);
tnc=length(clabel);
if S==1,
    Tt=[Tt ones(Q,1)-Tt];
    S=2;
end
for cn=1:tnc,
    tmpd=(Pt-ones(Q,1)*Cen(cn,:));
    tmpe=tmpd*nth(cinv,cn);
    Distan(cn,:)=sum(tmpe'.*tmpd');
end
edist=exp(-Distan);
llmat=[];
for i=1:S,
    idx=find(clabel==i);
    if(isempty(llmat)), llmat=ppri(idx)'*edist(idx,:);
    else, llmat=[llmat; ppri(idx)'*edist(idx,:)];
    end
end
[conf,ind]=max(llmat);
Idmat=eye(S);
class=Idmat(ind,:);
Cmat=Tt'*class;
Crate=100*sum(diag(Cmat))/Q;

5.2.3 mlgmm.m

function [W,Covar,priors]=mlgmm(x,c)
% Usage: [W,Covar,priors]=mlgmm(x,c)

[K,N]=size(x);
if nargin<1, c=0; end
er = 1e-5;
xmean=mean(x);
xstd=std(x);
itmax=min(K,30);
if nargin<2 | c == 0,
    covx=(x-ones(K,1)*xmean)'*(x-ones(K,1)*xmean)/K;
    [vx,dx]=eig(covx);
    dx0=diag(dx);
    [tmp, idx]=sort(-dx0);
    ex=dx0(idx);
    vx=vx(:,idx);
    c=sum((ex./max(ex))>.5);
end
if nargin<2 | c == 0,
    mix.centres=W;
    mix.covars=Covar;
    for i = 1:mix.ncentres
        tmp=reshape(mix.covars(i,:),N,N);
        if rank(tmp) < mix.nin
            tmp2 = tmp + GMM_WIDTH.*eye(mix.nin);
        end
        mix.covars(i,:) = tmp2(:)';
    end
    end
options(1) = -1;
options(14) = 1000;
options(5) = 1;
end
mix.centres;
Covar=mix.covars;
priors=mix.priors;

5.2.4 gmm.m

function mix=...
gmm(dim, ncentres, covar_type, ppca_dim)
% Copyright (c) Ian T Nabney (1996-9)
if ncentres < 1
    error(['Number of centres must be '...
          'greater than zero']);
end
mix.type = 'gmm';
mix.nin = dim;
mix.ncentres = ncentres;
mix.covar_type = covar_type;
mix.priors = ones(1,mix.ncentres) ./ mix.ncentres;
mix.centres = randn(mix.ncentres, mix.nin);
mix.covars = repmat(tmp(:)',mix.ncentres,1);
mix.nwts = mix.ncentres + mix.ncentres*mix.nin + ...
          mix.ncentres*mix.nin*mix.nin;

5.2.5 gmmactiv.m

function a = gmmactiv(mix, x)
errstring = consist(mix, 'gmm', x);
if ~isempty(errstring)
    error(errstring);
end
data = size(x, 1);
a = zeros(data, mix.ncentres);
for i = 1:mix.ncentres
    if(iscolumn(a(i,:)) & mixinga(i,1))
        a(i,:)=mixinga(i,1)/mixinga(i,2);
    end
end
W0=0.1*randn(c,N)+ones(c,1)*xmean;
[W,iter,Sw,Sb,Covar]=kmeansf(x,W0,er,imax);
mix = gmm(N, c, 'full');
N = sqrt(N2);
tmp = reshape(mix.covars(i,:,N,N));
c = chol(tmp);
temp = diffs/c;
a(:,i) = exp(-0.5*sum(temp.*temp, 2))./normal*prod(diag(c));
end

5.2.6 gmmem.m

function [mix, options, errlog] = gmmem(mix, x, options)

%Copyright (c) Ian T Nabney (1996-9)
errstring = consist(mix, 'gmm', x);
if ~isempty(errstring)
    error(errstring);
end
[ndata, xdim] = size(x);
if (options(14))
niters = options(14);
else
    niters = 100;
end

display = options(1);
store = 0;
if (nargout > 2)
    store = 1;
    errlog = zeros(1, niters);
end
test = 0;
if options(3) > 0.0
    test = 1;
end
check_covars = 0;
if options(5) >= 1
    if display >= 0
        disp('check_covars is on');
    end
    check_covars = 1;
    MIN_COUVAR = eps;
    init_covars = mix.covars;
end
disp('----- EM iterations commence -----')
best = inf; nstall = 0; maxstall = 20;
for n = 1:niters
    [post, act] = gmmpost(mix, x);
    if (display | store | test)
        prob = act*(mix.priors)';
        e = - sum(log(prob));
        if store
            errlog(n) = e;
        end
    fprintf(1, 'Cycle %4d Error %11.6f\n', n, e);
    end
    if test
        if (n > 1 & abs(e - eold) < options(3))
            options(8) = e;
            return;
        else
            eold = e;
        end
    end
    new_pr = sum(post, 1);
    new_c = post' * x;
    mix.priors = new_pr ./ ndata;
    mix.centres = new_c ./ (new_pr' * ones(1, mix.nin));
    for j = 1:mix.ncentres
diffs = x - (ones(ndata, 1) * mix.centres(j,:));
diffs = diffs.*(sqrt(post(:,j))*ones(1, mix.nin));
tmp = (diffs'*diffs/new_pr(j));
mix.covars(j,:) = tmp(:)';
end
if check_covars
    [c,N2] = size(mix.covars);
    N = sqrt(N2);
    for j = 1:mix.ncentres
        tmp = reshape(mix.covars(j,:),N,N);
        if min(svd(tmp)) < MIN_COUVAR
            mix.covars(j,:) = init_covars(j,:);
        end
    end
end
if e<best
    best=e;
    nstall=0;
else
    nstall=nstall+1;
end
if (nstall>maxstall)
    break
end
end
options(8) = -sum(log(gmmprob(mix, x)));
if (display >= 0)
    disp('Warning: Max iterations exceeded');
end

5.2.7 gmmpost.m

function [post, a] = gmmpost(mix, x)
errstring = consist(mix, 'gmm', x);
if ~isempty(errstring)
    error(errstring);
end
ndata = size(x, 1);
a = gmmactiv(mix, x);
post = (ones(ndata, 1)*mix.priors).*a;
s = sum(post, 2);
s = s + (s==0);
post = post./(s*ones(1, mix.ncentres));

5.2.8 gmmprob.m

function prob = gmmprob(mix, x)
% Check that inputs are consistent
errstring = consist(mix, 'gmm', x);
if ~isempty(errstring)
error(errstring);
end

% Compute activations
a = gmmactiv(mix, x);

% Form dot product with priors
prob = a * (mix.priors)';

5.2.9 kmeansf.m

function [W, iter, Sw, Sb, Cova]=kmeansf(X,W,er,itmax,tp)
% Usage: [W, iter, Sw, Sb, Cova]=
% kmeansf(X,W,er,itmax,tp)
% % kmeansf(X,W,er,itmax,tp)
% [c,N]=size(W);
% [K,N1]=size(X);
% if N~=N1, error(['W and X should have ' ...
% 'the same number of columns!']); end
if nargin < 5,
c=1;
itmax=c;
else
if nargin < 3,
er=0.01;
end
if nargin < 4,
itmax=c;
end
if nargin < 3,
iter=1;
end
end
dtype=tp(2);

if nargin < 4,
itmax=c;
end
if nargin < 3,
er=0.01;
end
if c==1,
iter=1; Sb=zeros(N); D=0;
if N==1,
W=X; Sw=0;
else
if N > 1,
W=mean(X);
end
tmp=dist(X,W,dtype);
end

return
end
converged=0; Dprevious=0; iter=0;
while converged==0,
5.2.10 kmeantest.m

```matlab
function [d,member,distance]=kmeantest(X,W)
% Usage: [d,member,distance]=kmeantest(X,W)
[c,N]=size(W);
[K,N]=size(X);
distance=dist(X,W);
[tmp1,ind]=sort(distance');
d=sqrt(sum(tmp1(1,:).*tmp1(1,:)));
member=ind(1,:');
end
```

5.2.11 dist.m

```matlab
function d=dist(X,W,type,para)
% Usage: d=dist(X,W,type,para)
if nargin <3,
    type=0;
end
if type == 3,
    if nargin < 4,
        error('must give covariance matrices'); end
end
[K,M]=size(X);
[C,M1]=size(W);
if M~=M1,
    error('X and W dimension should be the same!'); end
if type==0,
    if M >1,
       wnorm=sum((W.*W)');
    elseif M==1,
       wnorm=(W.*W)';
    end
    xnorm=sum((X.*X)');
    d=sqrt(xnorm'*ones(1,C)-
    2*X*W'+ones(K,1)*wnorm);
elseif type==1,
    d=[];
    if C <=K,
        for i=1:C,
            d=[d; max(abs(X(i,:)'*ones(1,C)-W'))];
        end
    else
        if type==2,
            d=[];
            if C <=K,
                for i=1:C,
                    d=[d; max(abs(X(i,:)'*ones(1,C)-W'))];
                end
            else
                if C <=K,
                    for i=1:C,
                        d=[d; max(abs(X(i,:)'*ones(1,C)-W'))];
                    end
                else
                    if C <=K,
                        for i=1:C,
                            d=[d; max(abs(X(i,:)'*ones(1,C)-W'))];
                        end
                    end
```

5.2.12 consist.m

```matlab
function errstring = ... 
consist(model, type, inputs, outputs)
%CONSIST Check that arguments are consistent.
errstring = '';
if ~isempty(type)
    if ~isfield(model, 'type')
        errstring = ['Data structure does not have type field'];
        return
    end
    s = model.type;
    if ~strcmp(s, type)
        errstring = ['Model type ', s,
            ' does not match expected type ', type, '
            does not match expected type '];
        return
    end
end
if nargin > 2
    if ~isfield(model, 'nin')
        errstring = ['Data structure does not contain nin field'];
        return
    end
    end
end
data_nin = size(inputs, 2);
if model.nin ~= data_nin
    errstring = ['Dimension of inputs ',
        ' does not match number of model inputs ',
        num2str(model.nin)];
    return
end
data_nin = size(inputs, 2);
if model.nin ~= data_nin
    errstring = ['Dimension of inputs ',
        ' does not match number of model inputs ',
        num2str(model.nin)];
    return
end
data_nin = size(inputs, 2);
if model.nin ~= data_nin
    errstring = ['Dimension of inputs ',
        ' does not match number of model inputs ',
        num2str(model.nin)];
    return
end
data_nin = size(inputs, 2);
if model.nin ~= data_nin
    errstring = ['Dimension of inputs ',
        ' does not match number of model inputs ',
        num2str(model.nin)];
    return
end
data_nin = size(inputs, 2);
if model.nin ~= data_nin
    errstring = ['Dimension of inputs ',
        ' does not match number of model inputs ',
        num2str(model.nin)];
    return
end
data_nin = size(inputs, 2);
if model.nin ~= data_nin
    errstring = ['Dimension of inputs ',
        ' does not match number of model inputs ',
        num2str(model.nin)];
    return
end
data_nin = size(inputs, 2);
if model.nin ~= data_nin
    errstring = ['Dimension of inputs ',
        ' does not match number of model inputs ',
        num2str(model.nin)];
    return
end
data_nin = size(inputs, 2);
if model.nin ~= data_nin
    errstring = ['Dimension of inputs ',
        ' does not match number of model inputs ',
        num2str(model.nin)];
    return
end
data_nin = size(inputs, 2);
if model.nin ~= data_nin
    errstring = ['Dimension of inputs ',
        ' does not match number of model inputs ',
        num2str(model.nin)];
    return
end
data_nin = size(inputs, 2);
if model.nin ~= data_nin
    errstring = ['Dimension of inputs ',
        ' does not match number of model inputs ',
        num2str(model.nin)];
    return
end
data_nin = size(inputs, 2);
if model.nin ~= data_nin
    errstring = ['Dimension of inputs ',
        ' does not match number of model inputs ',
        num2str(model.nin)];
    return
end
data_nin = size(inputs, 2);
if model.nin ~= data_nin
    errstring = ['Dimension of inputs ',
        ' does not match number of model inputs ',
        num2str(model.nin)];
    return
end
data_nin = size(inputs, 2);
if model.nin ~= data_nin
    errstring = ['Dimension of inputs ',
        ' does not match number of model inputs ',
        num2str(model.nin)];
    return
end
data_nin = size(inputs, 2);
if model.nin ~= data_nin
    errstring = ['Dimension of inputs ',
        ' does not match number of model inputs ',
        num2str(model.nin)];
    return
end
data_nin = size(inputs, 2);
if model.nin ~= data_nin
    errstring = ['Dimension of inputs ',
        ' does not match number of model inputs ',
        num2str(model.nin)];
    return
end
data_nin = size(inputs, 2);
if model.nin ~= data_nin
    errstring = ['Dimension of inputs ',
        ' does not match number of model inputs ',
        num2str(model.nin)];
    return
end
data_nin = size(inputs, 2);
if model.nin ~= data_nin
    errstring = ['Dimension of inputs ',
        ' does not match number of model inputs ',
        num2str(model.nin)];
    return
end
```
end
data_nout = size(outputs, 2);
if model.nout ~= data_nout
errstring = ['Dimension of outputs ',...
' does not match number of model outputs ',...
num2str(model.nout)];
return
end
num_in = size(inputs, 1);
um_out = size(outputs, 1);
if num_in ~= num_out
errstring = ['Number of input patterns ',...
' does not match number of output patterns ',...
num2str(num_out)];
return
end

5.3 KNN

5.3.1 knn.m

function [Cmat,Crate,class]=knn(Pr,Tr,Pt,Tt,kN)
% Usage: [Cmat,Crate]=knn(Pr,Tr,Pt,Tt,kN)
[K,N]=size(Pr);
[Q,S]=size(Tt);
rbias=sum(Pr'.*Pr')*.5;
if S==1,
oneS=eye(S+1);
Tt=[Tt ones(Q,1)-Tt];
Tr=[Tr ones(K,1)-Tr];
else
oneS=eye(S);
end
class=[];
for i =1:Q,
d=-Pr*Pt(i,:)+rbias';
[y,idx]=sort(d);
if kN > 1,
[yy,kidx]=max(sum(Tr(idx(1:kN),:)));
else
class=oneS(kidx,:);
end
else
class=[class;oneS(kidx,:)];
end
end

end % if kN > 1

end % i-loop

Cmat=Tt'class;
Crate=sum(diag(Cmat))*100/Q;

5.4 MOE

5.4.1 moe.m

clear
more off
format short
format compact
disp('Good Morning Dave');
disp('-----');
disp('Would you like to load a saved state?');
loadstate=input('enter 1 for yes, 0 for no: ');
if (loadstate==0)
oDim=input('how many classes do you have data for? ');
disp('What is the name of the data file?');
data_file=input('File name in single quotes: ');
eval(['load ' data_file]); data=eval(data_file);
eval(['clear ' data_file]);
[A E]=size(data);

end % if loadstate==0

dim=B-oDim;
trainData=data(:,1:iDim);
trainClass=data(:,iDim+1:B);
svtest=1;
else
num=ceil(0.2*A);
interval=ceil(A*rand(1));
for i=2:num
T(i)=max(mod(T(i-1)+interval,A),1);
end
T=sort(T);
[trainData,testData]=split(trainData,T);
[trainClass,testClass]=split(trainClass,T);
end % if loadstate==0

disp('I will load it after the device is trained');
svtest=input('enter 1 for yes, 0 for no: ');
if(svtest)
disp('what file name do you want me to use?');
svname=input('enter in single quotes: ');
test=[testData,testClass];
eval(['save -ascii ' svname ' test']);
end % if svtest
end % if testfile.
disp('would you like me to simplify the data?');
disp('depending on the data, it can help.');
simp=input('enter 1 for yes, 0 for no: ');
if (simp==1)
disp('----- Simplifying -----');
[trainData,rl]=simplify(trainData);
disp('----- Simplification Finished -----');
if (svtest==1)
disp('would you like the simplified data?');
svsimp=input('enter 1 for yes, 0 for no: ');
if (svsimp==1)
end % if svsimp
end % if svtest
else
rl=list(zeros(1,B),zeros(1,B));
end % if simp

[A B]=size(trainData);
disp('----- Scaling Data -----');
if(max(max(trainData))>100)
disp('your data has quite a range.');
disp('I put it on a log scale');
trainData=log(max(trainData,eps));
tooklog=1;
else
tooklog=0;
end % if max..
[trainData,dlow,dhigh]=scale(trainData,-5,5,1);
disp('----- Training MLP -----')
[bpCmat,bpCrate,bpW,bpAtype]=...
bptrain([trainData,trainClass],B);
disp('----- Training mlGMM -----')
[Cen,cinv,clabel,cc]=mltrnnew([trainData,trainClass]);
end % if loadstate
if(testfile==1)
disp('what is the name of the test file?');
test_file=input('File name in single quotes: ');
eval(['load ' test_file]); test=eval(test_file);
eval(['clear ' test_file]);
[C D]=size(test);
testData=test(:,1:iDim);
testClass=test(:,iDim+1:D);
testClass=[testClass'-ones(oDim,1)*...
max(testClass')==0'];
end

5.4.2 loadstate.m

function [bpW,cinv,rl,trainData,trainClass,...
bpCrates,bpAtype,Cen,clabel,...
nc,ppr,simp,dlow,dhigh,testfile,...
iDim,oDim,tooklog]=loadstate(name)
eval(['load ' name ' ;']);
bpW, cinv, rl, trainData, trainClass,
end % if svtest

else
disp('what is the name of the state file?');
svfile=editor('enter file name in single quotes: ');
[bpW,bpCrates,bpAtype,Cen,clabel,...
n,ppr,simp,dlow,dhigh,testfile,...
iDim,oDim,tooklog]=loadstate(name)
eval(['load ' name ' ;']);
bpW=editor(pi); rl=editor(pi); cinv=editor(pi);
for i=1:bpWL
eval(['bpW' int2str(i) ') = w...
5.4.3 savestate.m

function savestate(name,bpW,cinv,rl,trainData,...
trainClass,bpCrate,bpAtype,Cen,clabel,nc,...
ppri,simp,d1ow,d1igh,testfile,iDim,oDim,tooklog)

bpWL=length(bpW);
rlL=length(rl);
cinvL=length(cinv);

for i=1:bpWL
    eval(['w' int2str(i) ' = nth(bpW,'...
         int2str(i) ') ;']);
    varlist=[varlist ' w' int2str(i)];
end
for i=1:rlL
    eval(['rl' int2str(i) ' = nth(rl,'...
         int2str(i) ') ;']);
    varlist=[varlist ' rl' int2str(i)];
end
for i=1:cinvL
    eval(['cinv' int2str(i) ' = nth(cinv,'...
         int2str(i) ') ;']);
    varlist=[varlist ' cinv' int2str(i)];
end

eval(['save ' name ' ' varlist ';']);

5.4.4 scale.m

function [x,xmin,xmax]=scale(x,low,high,type)

if nargin==3,
type=0;
end %if

[K,M]=size(x);
range=high-low;
if type==0,
    xmax=max(x);
xrange=xmax-xmin;
x=(x-xmin)*range/xrange+low;
elseif type==1,
    if size(high)==size(low) == [1 1];
        high=ones(1,M)*high; low=ones(1,M)*low;
    end
end

low=diag(diag(low))'; range=diag(diag(range))';
xmax=max(x); xmin=min(x); xrange=xmax-xmin;
mask=[xrange<1e-3];
rangle=(1-mask).*range+mask;
xrange=(1-mask).*xrange+mask;
x=(x-ones(K,1)*xmin)*diag(range./xrange)+...
    ones(K,1)*low;
end

5.4.5 simplify.m

function [y,rl]=simplify(x,tol)

Usage [y redundant]=simplify(x,tol)

if nargin<2
    tol=0.001;
end %if

[h w]=size(x);
redundant=zeros(1,w);

disp('finding items with low mean');
for i=1:w
    if abs(mean(x(:,i)))<tol
        redundant(i)=1;
    end %if
end %for

disp('finding items with low stdev');
for i=1:w
    if (abs(std(x(:,i)))<tol)&(redundant(i)==0)
        redundant(i)=1;
    end %if
end %for

disp('finding equal vectors');
rs=zeros(1,w);
for i=1:w
    rs(i)=sum(x(:,i));
end
w2=w^2;
for i=1:w
    c=i*w;
    if(redundant(i)==0)
        for j=i+1:w
            if (abs(rs(i)-rs(j))<tol)&(redundant(j)==0)
                redundant(i)=1;
            end %if
        end %for j
    end %if redundant
end %for i

disp(["removing ', sum(redundant), ' items'");
x=remove(x,redundant);
rs=remove(rs,redundant);
[rh w]=size(x);

redundant=zeros(1,w);
disp(['finding 2-way linearly dependant '...
'vectors']);
w3=w^3;
for i=1:w
  c=i*w2;
disp(['processing ' int2str(c) ' of '...
'int2str(w3) ' (' int2str(100*c/w3) '%%)']);
  if(redundant(i)==0)
    for j=1:w
      if(redundant(j)==0)&&(i~=j)
        for k=1:w
          if (abs(rs(i)-rs(j)-rs(k))<tol)&...
             (redundant(k)==0)&&(j~=k)&&(i~=k)
            redundant(i)=1;
          end
        end
      end
    end
  end
end
disp(['removed ' int2str(sum(redundant))...
' items']);
rl{2}=redundant;
y=remove(x,redundant)

function y=remove(x,redundant)
%Usage: y=remove(x,redundant)
[h,w]=size(x);
tmp=zeros(h,w-sum(redundant));
index=1;
for i=1:w
  if redundant(i)==0
    tmp(:,index)=x(:,i);
    index=index+1;
  end
end
y=tmp;

function [data1,data2]=split(X,elements)
%Usage: [data1,data2]=split(X,elements)
[N K]=size(X);
snarf=zeros(1,N);
snarf(elements)=1;
data1=remove(X',snarf)';
data2=X(elements,:);

5.5 Matlab Compatibility

5.5.1 nth.m

function y=nth(l,x)
y=l{x};

5.5.2 list.m

function l=list(a,b,c,d,e,f)
switch nargin
case 1
  l={a};
case 2
  l={a b};
case 3
  l={a b c};
case 4
  l={a b c d};
case 5
  l={a b c d e};
case 6
  l={a b c d e f};
otherwise
  error('too many list args');
end
References

   http://dmoz.org/Science/Math/Software/MATLAB/

   http://carol.wins.uva.nl/~jverbeek/gm.html

   http://www-sig.enst.fr/~cappe/h2m/h2m.html

   http://www.nara.gov/fedreg/elctcoll/ecfront.html

   http://www.fec.gov/pages/tonote.htm

[6] The Presidential Vote Equation
   http://fairmodel.econ.yale.edu/vote/

[7] GNU Octave
   http://www.octave.org/