Multivariate Normals (MVN) Octave/Matlab Toolbox (Version 1)

Wednesday 31st August, 2011

Contents

1 The Toolbox

The MVN (MultiVariate Normal) Matlab/Octave toolbox implements divergences, centroids and algorithms (k-means, Self-Organizing Maps) to work with this non-vectorial of features. The toolbox is freely available on the Internet.[1].

Multivariate Gaussians are used in Music Similarity Algorithms, to represent timbre music features. Multivariate Gaussians and their attached Kullback-Leibler divergences are currently established as the de-facto standard method to compute music similarity. In this documentation we use Elias Pampalks music analysis (MA) Matlab toolbox [Pam04] to demonstrate how to use this toolbox (skip to Section 1.6 for some examples). Of course usage is not limited to music similarity models, any multivariate Gaussian features can be processed with this toolbox.

The centroid computing algorithms are implemented after Nielsen and Nock [NN09]. The k-means algorithms as it is described in Banjeree et al. [BMDG05], the Self-Organizing Maps algorithm for the Gaussians as in Schnitzer et al. [SFWG10].

[1] `http://www.ofai.at/~dominik.schnitzer/mnv`

1.1 Initialization

Initialization of MVN Gaussians is done using a single function call:

- `mvn1 = mvn_new(co, m)`. To instantiate a new n-dimensional multivariate Normal (MVN) object with the toolbox, pass the covariance matrix (`co`) and mean vector (`m`) of your Gaussian to the `mvn_new()` function. The return value `mvn1` is a Matlab structure with the following attributes:
  - `co`: The full $n \times n$ covariance matrix. If the matrix is not positive-definite an exception is thrown and execution aborted.
  - `m`: The n-dimensional mean vector of the Gaussian.
  - `ico`: For speed reasons we also pre-compute the inverse of the covariance matrix ($n \times n$) using the fast Cholesky-decomposition.
  - `logdet`: We store the logarithm of the determinant of the covariance matrix `co`. `logdet` is computed from the Cholesky-decomposition.

  To estimate a n-dimensional Gaussian from a $m \times n$ data matrix (like a matrix of MFCC vectors) just use the built-in Matlab functions **`cov`**`(data)` and **`mean`**`(data)`.

1.2 Divergences

The toolbox implements a variety of divergences which can be used to compute a distance or similarity between two MVN models.

- `d = mvn_div_kl(m1, m2)`

  Computes the (asymmetric) Kullback-Leibler divergence between the MVN `m1` and `m2`.

- `d = mvn_div_skl(m1, m2)`

  Computes the symmetric Kullback-Leibler divergence between the MVN `m1` and `m2`.

- `d = mvn_div_js(m1, m2)`

  Computes the Jensen-Shannon-like divergence between MVN `m1` and `m2`. Since it is not possible to compute the Jensen-Shannon divergence between two multivariate Normals, we use an approximation which works quite well in our approximations (cf. [PSS+09]).

- `D = mvn_divmat(models, divergence)`

  This is a meta-function to quickly compute the whole similarity matrix for the given `models` and `divergence`. The parameter `models` is a struct-array of MVN models and `divergence` is a string specifying the divergence to use: `'kl_left'`, `'kl_right'`, `'skl'`, `'js'`. The full distance matrix `D` is returned in single precision to save memory.

1.3 Centroids

- `c = mvn_bregmancentroid_kl_left(models)`

  Computes the left-sided Kullback-Leibler centroid of the given MVN models.

- `c = mvn_bregmancentroid_kl_right(models)`

  Computes the right-sided Kullback-Leibler centroid of the given MVN models.

- `c = mvn_bregmancentroid_kl_skl(models, approx)`

  Computes the symmetrized Kullback-Leibler centroid of the given MVN models. If the parameter `approx` is not set, the centroid is computed using a geodesic walk algorithm. If `approx` is set to the value 1, an approximative centroid is returned. The approximative centroid is computed very fast, and is in many cases sufficiently exact.

1.4 Clustering

- `[centers, assig, qe] = mvn_kmeans(models, k, divergence)`

  This function implements the k-means clustering algorithm for MVN models. It does that in regard to the selected divergence. Similar to the `mvn_divmat()` function the `divergence` parameter selects the divergence to use for the k-means clustering: `'kl_left'`, `'kl_right'`, `'skl'`, `'skl_mid'`.

  Return values are a struct-array with the centers in `centers`, a vector which assigns each MVN to a centroid (`assig`) and a quantization error (`qe`) according to the divergence chosen.

- `[mapunits D] = mvn_som_skl(models, n)`

  This function trains a basic self-organizing map with the generalized SOM algorithm (as defined in [SFWG10]). The dimensions are specified by the parameter `n` which in turn creates a square SOM with the dimension $n \times n$. To compute the SOM the function uses the MVN models in parameter `models` and the divergence selected by `divergence`. The divergence paramter can be one of these divergences: `'kl_left'`, `'kl_right'`, `'skl'`, `'skl_mid'`.

  The first return value `mapunits` is a $n \times n$ SOM grid where each map unit is represented by an MVN model. The whole SOM has n^2 map units. The structure array has the following structure:

  - `x, y` the x/y-axis position of the map unit
  - `n` An array which stores the indices of the models which are assigned to this map unit.

  The return value `D` is an $(n * n) \times m$ matrix, where m is the number of `models` used during computation.

1.5 Additional Functions

- `h = mvn_entropy(m1)`

  Computes the entropy of the given an N-dimensional MVN `m1` with the covariance Matrix Σ. The entropy h is computed as:

  $$h = \frac{1}{2}\left(N + N\ln\left(2\pi\right) + \ln\left|\Sigma\right|\right) \tag{1}$$

- `p = mvn_ismetric(D)`

  Given a divergence matrix `D`, returns the fraction of triples elements obeying the triangle inequality.

1.6 Usage Examples

Feature Extraction We use the Music Analysis (MA) Matlab toolbox[2] by Elias Pampalk to extract audio music similarity features for the ISMIR2004 *Genre/Artist Identification/Classification* collection which is available freely on the web[3]. To do so we extract the features with the command:

```
ma_g1c_FeatureExtraction('ismir04_filelist.txt', 'features_dir/');
```

The the text file `ismir04_filelist.txt` lists all filenames of the ISMIR 2004 dataset introduced before. The above command does the feature extraction for all songs given in the text file and writes the features in the specified directory.

Loading the Features To load the features which, we first load the features from the "G1C_features.mat" file to memory:

```
load features_dir/G1C_features.mat
```

After that we prepare the features for MVN processing and load the filenames which encode the music genre in their path.

```
models = mvn_new(squeeze(data.feat.g1.co(1,:,:)),...
                 squeeze(data.feat.g1.m(1,:,:)));

for i = 2:length(data.filenames)
  models(i) = mvn_new(squeeze(data.feat.g1.co(i,:,:)),...
                      squeeze(data.feat.g1.m(i,:,:)));
end

filenames = importdata('ismir04_files.txt');
[genres grene_assignment] = mvn_fn2class(filenames, 4, '/');
```

[2] http://www.pampalk.at/ma/

[3] http://ismir2004.ismir.net/genre_contest/index.htm

Lines 1-7 initialize the MVN models, line 9-10 loads the filenames and extracts the genre names from the filenames. We will use the genres of the individual files for classification experiments later on.

Similarity and K-Means Clustering In this step we compute a divergence matrix using the Symmetric Kullback-Leibler divergence (line 1). For completeness and to check if everything was done correctly, we compute the 1-nearest neighbor classification accuracy (line 2, `nn1_accuracy = 0.7819` in the example).

```
1    D = mvn_divmat(models, 'skl');
2    nn1_accuracy = mvn_knnclass(D, genre_assignment, 1);
3
4    [centroids c_assignment] = mvn_kmeans(models, 10, 'skl');
```

In line 4 we compute a randomly initialized k-means clustering of our MVN models. In the return value `centroids` we return the 10 centroids found with the k-means clustering. The variable `c_assignment` stores the index of the centroid a MVN model is assigned.

Figure 1 displays the cluster/music genre assignment confusion matrix which is generated from the k-means clustering. We clustered the collection into 10 clusters. In the figure it can be seen that the clustering which was emerging has a jazz/blues cluster (3), multiple classical/world clusters (1, 2, 4, 5, 6, 8), a strong metal/punk cluster (9), a pop/rock/electronic cluster (7) and an electronic/pop/rock/jazz cluster (10).

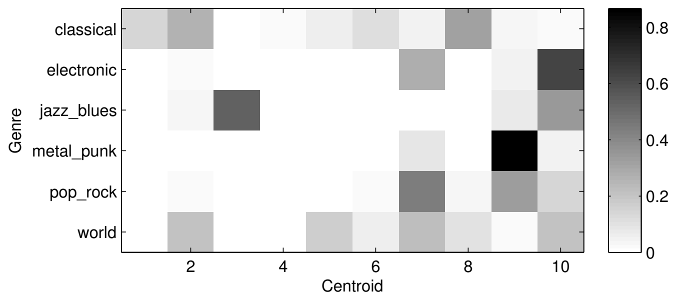

Figure 1: Confusion matrix displaying the genre to k-means cluster assignment. The various clusters with their relative genre composition can be seen. An optimal clustering

Computing and visualizing a SOM In the next code snippet we compute the SOM and their labels.

```
    [som_grid, som_D] = mvn_som_skl(models, [20 20]);
```

```
2
3      % Compute the labels for the SOM
4      labels = cell(length(som_grid), 1);
5      for i=1:length(som_grid)
6          if (isempty(som_grid(i).n)) continue; end
7          ng = genre_assignment(som_grid(i).n);
8          check = unique(ng);
9          maxct = 0; maxj = 0;
10         for j=1:length(check)
11             nct = length(find(ng == check(j)));
12             if (nct > maxct)
13                 maxj = j;
14                 maxct = nct;
15             end
16         end
17         if (maxct < 3) continue; end
18         labels{i} = genres{check(maxj)};
19     end
```

Finally we visualize the SOM using the Smoothed-Data Histograms Matlab Toolbox[4] and label it according to the genre label names we just prepared in the last code snippet (line 4-19)

```
M.dist_codebook = 1:(20*20);
M.topol.msize = [20 20];
S = sdh_calculate(som_D, M, 'spread', 10);

% Visualize SOM using the SDH Toolbox
sdh_visualize(S, 'sofn', 0, 'labels',  labels);
```

Figure 2 shows the Matlab Figure displaying the SOM using very rudimentary music similarity features. The Smoothed-Datagram visualization which was first presented for Music Collections in [Pam03] is called *Islands of Music*. When looking at the map we can see that large *Classical Music* islands emerged. On the bottom of the visualization we can see an *Electronic Music* island which is connected to a *Metal* island. On top of the map there is a *World Music* island.

[4]http://www.ofai.at/~elias.pampalk/sdh/

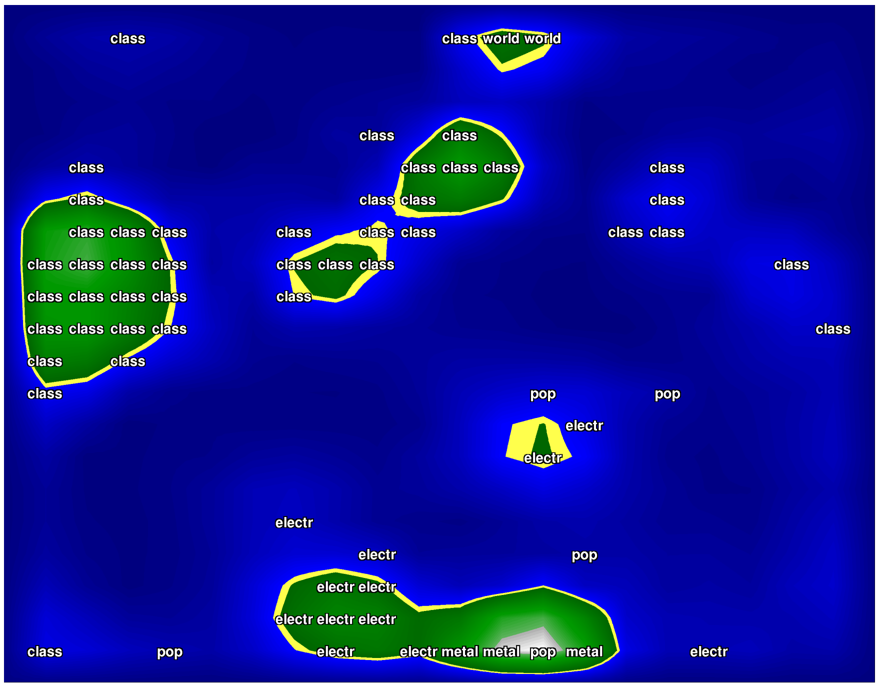

Figure 2: *Islands Of Music* visualization of the ISMIR 2004 collection using standard timbre music similarity features extracted with the MA Matlab toolbox.

1.7 License

References

[Pam04] E. Pampalk. A matlab toolbox to compute music similarity from audio. In *Proceedings of the 5th International Conference on Music Information Retrieval. ISMIR'04*, pages 254–257, 2004.

[BMDG05] A. Banerjee, S. Merugu, I. S. Dhillon, and J. Ghosh. Clustering with bregman divergences. *The Journal of Machine Learning Research*, 6:1705–1749, 2005.

[NN09] F. Nielsen and R. Nock. Clustering multivariate normal distributions. *Emerging Trends in Visual Computing*, pages 164–174, 2009.

[SFWG10] D. Schnitzer, A. Flexer, G. Widmer, and M. Gasser. Islands of Gaussians: The Self Organizing Map and Gaussian Music Similarity Features. In *Proceedings of the 11th International Conference on Music Information Retrieval. ISMIR'10*, 2010.

[PSS+09] T. Pohle, D. Schnitzer, M. Schedl, P. Knees, and G. Widmer. On rhythm and general music similarity. In *Proceedings of the 10th International Conference on Music Information Retrieval. ISMIR'09*, 2009.

[Pam03] E. Pampalk. Islands of music: Analysis, organization, and visualization of music archives. *Journal of the Austrian Society for Artificial Intelligence*, 22(4):20–23, 2003.