INFO411 Lecture 3 - Online Clustering

Jeremiah Deng

University of Otago

25/7/2017

Overview

References

- Alpaydin, Chapter 12 (2nd / 3rd Ed.)
- Hertz, Krogh & Palmer, *Introduction to the Theory of Neural Computation*, Chapter 9
- Duda, Hart & Stork, *Pattern Classification*, Section 10.11

The k-Means Algorithm

- Given k, the k-means algorithm is implemented in the following steps:
  1. Partition data items into k non-empty subsets
  2. Obtain the centroids as the centers (mean points) of the partitions.
  3. Obtain new partitions: assign each data item to the cluster of the nearest centroid.
  4. Stop when no more new assignment is found; otherwise go back to Step 2.
- The algorithm may need to go through many iterations before it terminates or converges.

Online Learning: Challenges

- In traditional clustering,
  - Cluster structure can be sensitive to small changes or noises in data.
  - Clustering is mostly done in batch mode.
- In online learning:
  - Data may arrive incrementally but constantly
  - Limited memory: data need to go through single-pass
  - Limited processing time
  - Evolving data: concept drifts may exist
- What's required:
  - **Incremental learning** ability: learning data piece-by-piece.
  - **Stability**: cluster structure not easily drifted
  - **Plasticity**: being adaptive and possibly allowing new clusters

Online averaging: a bit of DIY

$n \leftarrow 0, \text{avg}_0 \leftarrow 0$
while true **do**
  $x_n \leftarrow \text{random}()$
  $\text{avg}_n \leftarrow \text{avg}_{n-1} + \gamma_n(x_n - \text{avg}_{n-1})$
  $n \leftarrow n + 1$
end while

Good experimental results can be obtained with very small γ values, or $\gamma_n = 1/n^p$, $p > 1$.

In real-world scenarios with dynamic data environments does this work? *Let's find out...*

Optimization in Online k-Means

Another take on the reconstruction error for k-means clustering:

$$E(\{\mathbf{m}_i\}_{i=1}^k|\mathcal{X}) = \sum_t \sum_i b_i^t \|\mathbf{x}^t - \mathbf{m}_i\|^2$$

where

$$b_i^t = \begin{cases} 1 & \text{if } \|\mathbf{x}^t - \mathbf{m}_i\|^2 = \min_j \|\mathbf{x}^t - \mathbf{m}_j\| \\ 0 & \text{otherwise} \end{cases}$$

In online learning, we approximate gradient descent with stochastic gradient descent (SGD), doing a small update on clusters at each step. The criterion function at step t is

$$E^t(\{\mathbf{m}_i\}_{i=1}^k|\mathbf{x}^t) = \sum_t \sum_i b_i^t \|\mathbf{x}^t - \mathbf{m}_i\|^2$$

By SGD (see e.g. (Bottou & Benjio, 1995)), we have

$$\Delta\mathbf{m}_i = -\eta \frac{\delta \mathrm{E}^t}{\delta \mathbf{m}_i} = \eta b_i^t(\mathbf{x}^t - \mathbf{m}_i)$$

Competitive Learning

- Competitive learning is a methodology based on neuroscience research.
- CL schemes
  - Basic competitive learning
    - Fixed number of clusters
    - "Winner-takes-all"
  - Soft competitive learning
    - Allows multiple winning neurons
  - Leader-Follower clustering
    - Allows a variable number of neurons

Basic C.L. algorithm

- a.k.a. 'local k-means'

Pseudocode

1. Initialize weights $\{\mathbf{w}_i\}, i = 1, 2, ..., k$
2. Randomly select a pattern $\mathbf{x}$
3. Find the winner neuron:
   $b = \text{argmin}\|\mathbf{x} - \mathbf{w}_i\|$
4. Update the winner neuron
   $\Delta\mathbf{w}_b = \gamma(\mathbf{x} - \mathbf{w}_b)$
5. Goto step 2 until no significant change in weights.

CL Characteristics

- Localized learning - good for online implementation
- Local minimum problem
- Fixed number of neurons
- Slow adaptability to novelty
  - Can you tell why?

CL: How to improve?

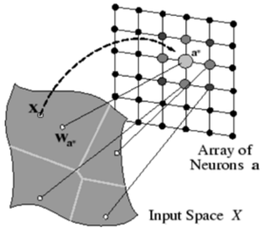
- Instead of tuning the winning neuron alone, other neurons also involved in adapting?
  - More robustness in the 'codebook'.
  - Can introduce relationship between prototypes.
  - However more time-consuming
- Dealing with uneven winning frequencies: frequency-sensitive FSCL, rival penalty RPCL
- More adaptability? E.g.,
  - growing and pruning,
  - merging and splitting etc.
- Can the learnt prototypes be useful for classification?
- Parallel implementation?

Self-Organizing Maps

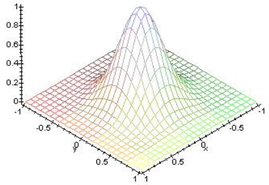
- Kohonen (1982)
- aka Self-organizing feature map (SOFM) or Kohonen map
- Found thousands of applications, including:
  - Speech recognition
  - Image compression
  - Bankruptcy prediction
  - Telecommunication traffic monitoring
  - Process control
  - Web document indexing

The SOM Model

- Introduces a topology for prototype nodes (ordering, neighbourhood)
- Define a neighbourhood function $\Omega(y_i, y_b)$ for prototype indeces $\{y_i\}$:
  - Bubble: $\Omega(y_i, y_b) = 1$ or 0
  - Gaussian: centered at the winner
  - "Mexican hat": lateral inhibition
- Nodes within the neighbourhood of the winner also get updated.



The nodes arranged in 2xD grids



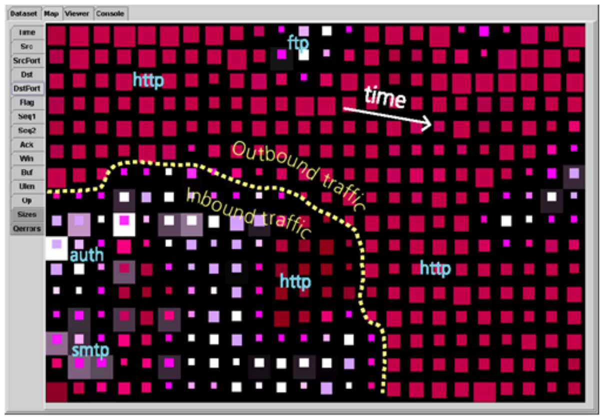
The 'Mexican hat' neighbourhood

The SOM algorithm

- Each adaptation tunes the winner (or "best matching unit" / BMU) and its neighbours:
  $\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \gamma(t)\Omega(y_i, y_b)(\mathbf{x} - \mathbf{w}_i)$
- During the iterations
  - Neighbourhood $\Omega(y_i, y_b)$ shrinks over time
  - Learning rate $\gamma(t)$ reduces over time
- Can operate either incrementally, or in batch mode

Example 1: Packet monitoring

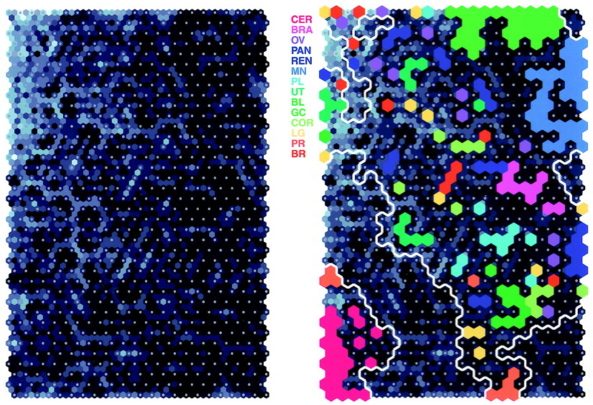
Mapping multi-dimensional packet data, one can use SOM to analyze network traffic, monitor online traffic, or even visualize intrusions.



Luc Girardin, USENIX'99 workshop

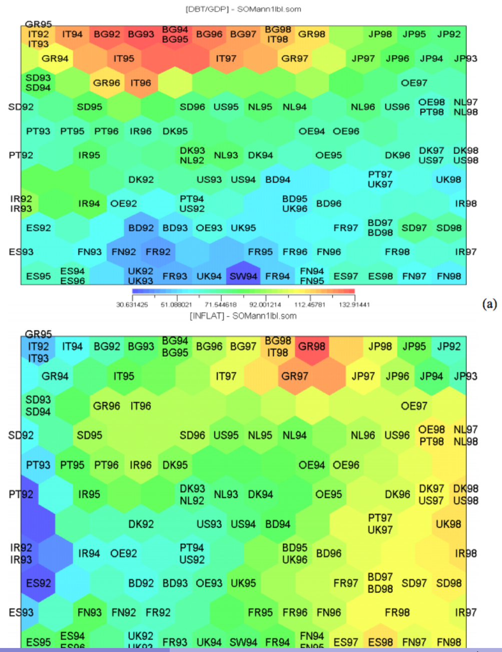
Example 2: Microarray data classification

Covet et al., Molecular Classification of Cancer: Unsupervised Self-Organizing Map Analysis of Gene Expression Microarray Data, *Mol Cancer Ther*, March 2003 2; 317



Example 3: EMU macroeconomics analysis

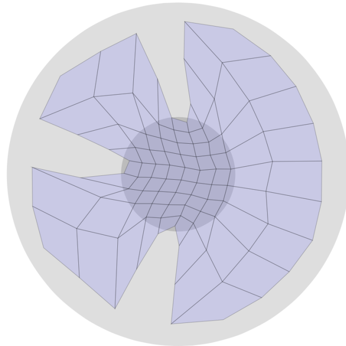
- Kasabov et al. (2001)
- Macroeconomic data collected for then EMU countries and non-EU countries
- Attributes include: GDP, debt, deficit, inflation rate, interest rate, unemployment, balance of payment, production gap etc.
- Trained SOMs can be coloured by components



SOM: Characteristics

Positives:
- Multi-dimension scaling (often onto 2-D)
- Probability density approximation: more ‘prototypes’ generated for regions of higher probability densities.
- Topology preserving: any two close input patterns should match to the same neuron, or two neurons in a neighbourhood on the map.

Negatives:
- Rigid map topology
- Fixed number of units
- Limited online learning ability



DemoGNG results on “Fovea”

Neural Gas

- Martinetz (1993)
- Topology constraint in SOM removed ☺
- Prototypes organised in the original space
- Weight updating rule: $\Delta w_i = \gamma h(k_i)(\mathbf{x} - \mathbf{w}_i)$
  - k_i: neighbour rank of the prototypes
  - E.g. for winner, $k_i = 1$; second winner $k_i = 2$ etc.
  - $h(k_i(\mathbf{x};\mathbf{w})) = e^{-k_i(\mathbf{x};\mathbf{w})/\lambda}$
- Neighbour ranking is time-consuming ☹
- Fixed number of neurons ☹

Leader-Follower

- Model itself is incremental; allows adaptive clustering without a known number of clusters
- Needs a similarity threshold (vigilance) or a distance threshold T
- This threshold implicitly controls the number of prototypes generated
- Procedure:
  1. Take initial inputs as prototypes (leaders)
  2. Modify existing prototypes with new input if they are similar (followers)
  3. Otherwise add the new input as a new prototype
  4. Repeat Steps 2-3 on new arriving data

Leader-Follower Algorithms

Pseudocode

\# *Assign first input to node 1*
$\mathbf{w}_1 \leftarrow \mathbf{x}$
\# *Number of nodes set as 1*
$K = 1$
while more data are available
  accept new $\mathbf{x}$
  $b \leftarrow \arg\min_i \|\mathbf{x} - \mathbf{w}_i\|$ # *find best match unit*
  if $\|\mathbf{x}\text{-}\mathbf{w}_b\| < T$ # *if close enough, update BMU*
    modify $\mathbf{w}_i$
  else # *otherwise insert as new*
    $K \leftarrow K + 1$
    $\mathbf{w}_K \leftarrow \mathbf{x}$
  endif

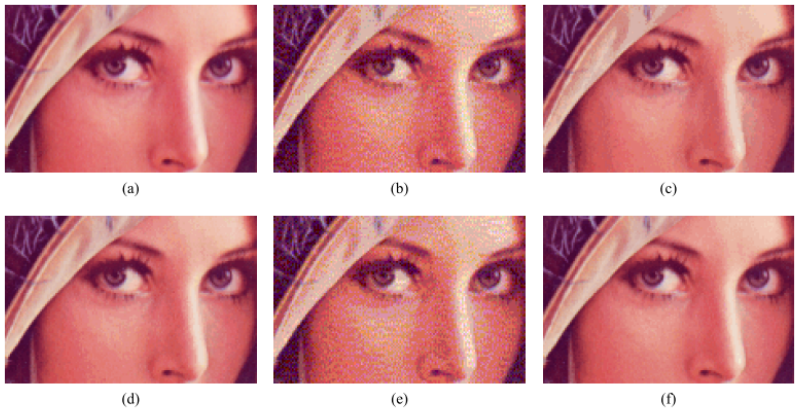
ART algorithms

- An implementation of L.F. algorithm
- Carpenter & Grossberg (1987). Adaptive resonance theory is to model how biological neural networks coping with novel patterns.
- Uses a vigilance parameter
- A family
  - ART1 for binary patterns
  - ART2/ART3 for analog patterns
  - ARTMAP as a supervised model
  - Fuzzy ARTMAP as a fuzzy variation

Evolving SOM

- Remove rigid 'grid' topology for the nodes (borrowing GNG's idea).
- Starts from a null network.
- Nodes establish connections by searching out two nearest neighbours.
- If input is far away from any nodes (using a distance threshold, similar to GNG ☹), add a node;
- Otherwise, update nodes within the neighbourhood according to their activation (a_i) stimulated by the input:
  - $\Delta\mathbf{w}_i = h_i(\mathbf{x})(\mathbf{x} - \mathbf{w}_i)$
  - Weighting, not ranking as in NG: $h_i(x) = \dfrac{a_i}{\sum_k a_k}$
  - The closer $\mathbf{x}$ is to $\mathbf{w}_i$, the bigger a_i

Example: Image colour quantization



True-colour images quantized into 256 colours.

(a) Original, (b) Median cut, (c) Octree, (d) Wu's method, (e) Local K-means, (f) ESOM.

Example: Online EM for background modeling

- Problem: monitor pixel changes in a video frame and separate foreground from background
- Solution (Stauffer & Grimson CVPR'99):
  - Probabilistic model for separating the background and foreground.
  - Adaptive mixture of multi-modal Gaussians per pixel.
  - Method for updating the Gaussian parameters.
  - Heuristic for determining the background.

The Adaptive MoG Model

- Each pixel is modelled by a mixture of K Gaussian distributions:

$$\eta(\mathbf{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}_k|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1}(\mathbf{x}-\boldsymbol{\mu}_k)}$$

$$\boldsymbol{\Sigma}_k = \sigma_k \mathbf{I}$$

- Look for Gaussians winning the most with the least variance; order models by w_i/σ_i
- The first B distributions are used as a model of the background (T is a threshold):

$$B = \text{argmin}_b \left(\sum_{j=1}^{b} w_j > T\right)$$

Learning the MoG

- If Model k matched to the current pixel value at time t, update its weight (α is a learning rate):

$$w_{k,t} = (1-\alpha)w_{k,t-1} + \alpha$$

- Updating the matched model:

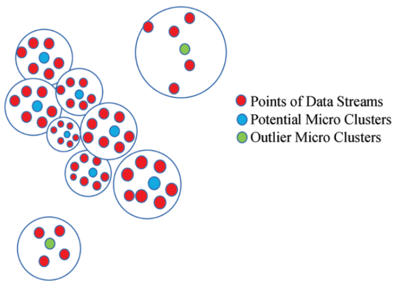
$$\mu_t = (1-\rho)\mu_{t-1} + \rho X_t$$

$$\sigma_t^2 = (1-\rho)\sigma_{t-1}^2 + \rho(X_t - \mu_t)^T(X_t - \mu_t)$$

where $\rho = \alpha\eta(X_t|\mu_k, \sigma_k)$

Stream Clustering

- Mining massive, unbounded sequences of data objects of rapid but often changeable rates.
- Example applications: Sensor networks, smart homes, Internet traffic monitoring, ATM transactions ...
- Approaches: partition (ClusStream), grid-based (DStream), density-based (DenStream)
- Tools: MOA, RapidMiner etc.
- Challenges: concept drift



Mini-batch k-means

- Sculley, Web-Scale K-Means Clustering, WWW'10
- Mini-batches tend to have lower stochastic noise than individual examples in SGD

Algorithm 1 Mini-batch k-Means.

Given: k, mini-batch size b, iterations t, data set X
Initialize each $\mathbf{c} \in C$ with an $\mathbf{x}$ picked randomly from X
$\mathbf{v} \leftarrow 0$
for $i = 1$ to t **do**
  $M \leftarrow b$ examples picked randomly from X
  for $\mathbf{x} \in M$ **do**
    $\mathbf{d}[\mathbf{x}] \leftarrow f(C, \mathbf{x})$ // Cache the center nearest to $\mathbf{x}$
  end for
  for $\mathbf{x} \in M$ **do**
    $\mathbf{c} \leftarrow \mathbf{d}[\mathbf{x}]$ // Get cached center for this $\mathbf{x}$
    $\mathbf{v}[\mathbf{c}] \leftarrow \mathbf{v}[\mathbf{c}] + 1$ // Update per-center counts
    $\eta \leftarrow \frac{1}{\mathbf{v}[\mathbf{c}]}$ // Get per-center learning rate
    $\mathbf{c} \leftarrow (1 - \eta)\mathbf{c} + \eta\mathbf{x}$ // Take gradient step
  end for
end for

Recap

- The online averaging problem
- Competitive learning: online k-means
- Other online algorithms
- Leader-follower
- Density-based
- ☺ Your algorithm?

Further Readings

- ER3: Kaur et al., Stream clustering algorithms: a primer, in *Big Data in Complex Systems*, 105-145, 2015.
- ER4: Lühr and Mihai Lazarescu. 2009. Incremental clustering of dynamic data streams using connectivity based representative points. *Data Knowl. Eng.* 68.
- Silva et al., Data stream clustering: A survey, *ACM Computing Surveys*, 46:1, DOI: 10.1145/2522968.2522981.
- Cao et al., Density-based clustering over an evolving data stream with noise, SDM'06, DOI: 10.1137/1.9781611972764.29.
- DemoGNG, URL http://www.demogng.de