LELEC2870: séance d'exercices 2
Vector quantization

9 novembre 2012

1 Objectives

Au cours de cette seconde séance d'exercices, vous devrez implémenter plusieurs algorithmes de vector quantization : le competitive learning, le frequency sensitive learning et le neural gas. La quantization consiste à réduire la taille du dataset tout en minimisant la perte d'information, en résumant celui-ci à l'aide de centroïdes. Les données et un script SciLab sont disponibles dans l'archive fournie.

2 Vector quantization

Considérons un ensemble de P vecteurs $\{\mathbf{x}_p | p = 1...P\}$ dans un espace à D dimensions. Une vector quantization consiste en un ensemble de Q vecteurs $\{\mathbf{y}_q | q = 1...Q\}$ de dimension D qui minimisent un critère d'erreur lié à la perte d'information. Généralement, Q est bien plus petit que P, puisque le but de la vector quantization est de résumer un grand nombre d'instances en utilisant seulement quelques points bien choisis. Ces vecteurs sont appelés *centroïdes* et un ensemble de centroïdes est appelé un *codebook*.

Généralement, le critère d'erreur est défini en termes de distance entre les données et les centroïdes. Souvent, il s'agit de minimiser la distance moyenne entre les données et les cnetroïdes

$$\mathbf{E} = \frac{1}{P} \sum_{i=1}^{P} \|\mathbf{x}_p - \mathbf{y}_{q_p}\|^2 \tag{1}$$

où q_p est l'index du plus proche centroïde pour le point $\mathbf{x}_p$.

3 Competitive learning

Le competitive learning est un algorithme de type *winner-take-all*, c'est-à-dire qu'à chaque présentation d'une instance $\mathbf{x}_p$, seulement le centroïde gagnant est bougé. Le centroïde $\mathbf{y}_q$ est dit *gagnant* si il est le plus proche du point $\mathbf{x}_p$ présenté, par rapport à une mesure de distance d. En d'autres termes,

$$\mathbf{y}_q \text{ gagne si } \forall r : d(\mathbf{x}_p, \mathbf{y}_q) \leq d(\mathbf{x}_p, \mathbf{y}_r). \tag{2}$$

La *règle d'adaptation* du codebook est alors définie par

$$\mathbf{y}_q \leftarrow \mathbf{y}_q + \alpha_k \left[\mathbf{x}_p - \mathbf{y}_q\right] \tag{3}$$

où $\alpha_k > 0$ est un facteur qui contrôle la vitesse du processus d'adaptation. Afin d'obtenir une convergence, le facteur α_k doit décroître au fur et à mesure des époques pour finalement devenir proche de zéro. Une méthode courante est la décroissance hyperbolique

$$\alpha_{k+1} \leftarrow \frac{\alpha_k \beta}{\alpha_k + \beta} \tag{4}$$

où β est une constante. Une époque prend fin après que chaque observation ait été utilisé. En général, il est conseillé de rendre l'ordre de présentation des données à chaque époque aléatoire.

Implémentez le competitive learning en utilisant deux méthodes d'initialisation du codebook, c'est-à-dire en sélectionnant les centroïdes au hasard dans l'espace ou aléatoirement parmi les observations. Visualisez les vector quantizations obtenues grâce à la fonction *show_quantization*. Quel problème rencontrez-vous avec la première méthode d'initialisation ? Est-ce également le cas avec la seconde ?

4 Frequency sensitive larning

Le frequency sensitive learning (FSL) est également un algorithme de type winner-take-all. Toutefois, en introduisant une pénalisation pour les centroïdes qui gagnent souvent, le problème des centroïdes perdus est évité. Un centroïde est dit *perdu* lorsqu'il n'est jamais choisi comme gagnant.

La règle de sélection du centroïde gagnat est remplacée par

$$\mathbf{y}_q \text{ gagne si } \forall r : u_q d\left(\mathbf{x}_p, \mathbf{y}_q\right) \leq u_r d\left(\mathbf{x}_p, \mathbf{y}_r\right) \tag{5}$$

où u_q est lié à la fréquence à laquelle $\mathbf{y}_q$ est sélectionné. Ce facteur u_q est initialement égal à 1 et est incrémenté chaque fois que le centroïde $\mathbf{y}_q$ est sélectionne comme gagnant. La règle d'adaptation reste

$$\mathbf{y}_q \leftarrow \mathbf{y}_q + \alpha_k \left[\mathbf{x}_p - \mathbf{y}_q\right]. \tag{6}$$

Implémentez le FSL. Comparez à nouveau les deux stratégies d'initialisation du codebook et vérifiez que le problème du centroïde perdu est résolu. Pourquoi en est-il ainsi ?

5 Neural gas

Contrairement aux deux algorithmes précédents, le neural gas est un algorithme de type *winner-take-most*, ce qui signifie que tous les centroïdes sont adaptés à chaque présentation d'un vecteur $\mathbf{x}_p$. Cette adaptation dépend de la distance entre le centroïde et $\mathbf{x}_p$.

La première étape de l'algorithme consiste à trier les centroïdes en fonction de leur distance (euclidienne ou autre) par rapport à $\mathbf{x}_p$. Ensuite, on peut définir une fonction de voisinage h telle que, si $\mathbf{y}_q$ est le $c_{\text{ème}}$ plus proche centroïde par rapport à $\mathbf{x}_p$, alors

$$h\left(\mathbf{x}_p, \mathbf{y}_q\right) = c - 1. \tag{7}$$

Notez que cette quantité est nulle pour le centroïde le plus proche ($c = 1$).

La seconde étape consiste à adapter chaque centroïde en utilisant la règle

$$\mathbf{y}_q \leftarrow \mathbf{y}_q + \alpha_k \exp\left(-\frac{h\left(x_p, \mathbf{y}_q\right)}{\lambda}\right)\left[\mathbf{x}_p - \mathbf{y}_q\right] \tag{8}$$

où λ régule la quantité de voisins qui sont affectés par la mise à jour. Ce facteur peut décroître au cours des époques.

Implémentez le neural gas. Expliquez l'influence du paramètre λ. Que se passe-t-il quand $\lambda = 0$? Et quand $\lambda \to +\infty$? Quel est l'avantage d'une stratégie de type winner-take-most par rapport à une stratégie de type winner-take-all ?