ESANN'2009 proceedings, European Symposium on Artificial Neural Networks - Advances in Computational Intelligence and Learning. Bruges (Belgium), 22-24 April 2009, d-side publi., ISBN 2-930307-09-9.

# Kernelizing Vector Quantization Algorithms

Matthieu Geist[1,2,3] and Olivier Pietquin[1] and Gabriel Fricout[2]

1- Supélec, IMS Research Group, Metz, France

2- ArcelorMittal Research, MC Cluster, Maizières-lès-Metz, France

3- INRIA Nancy - Grand Est, CORIDA project-team, France

**Abstract**. The kernel trick is a well known approach allowing to implicitly cast a linear method into a nonlinear one by replacing any dot product by a kernel function. However few vector quantization algorithms have been kernelized. Indeed, they usually imply to compute linear transformations (*e.g.* moving prototypes), what is not easily *kernelizable*. This paper introduces the Kernel-based Vector Quantization (KVQ) method which allows working in an approximation of the feature space, and thus kernelizing any Vector Quantization (VQ) algorithm.

## 1 Introduction

A common approach to handle nonlinear problems is to map the initial data set to a (generally higher dimensional) feature space which preserves the inherent data groupings and in addition simplifies the associated structure of data. However, as this feature space may be of high and possibly infinite dimension, directly working with the transformed variables is generally considered as an unrealistic option. This is the aim of the kernel trick described hereafter.

A kernel $K$ is a continuous, symmetric and positive semi-definite function. The kernel trick relies on the Mercer theorem [1] which states that each kernel can be expressed as a dot product in a higher dimensional space. More precisely, for each kernel $K$, there exists a mapping $\varphi : \mathcal{X} \subset \mathbb{R}^n \rightarrow \mathcal{F}$ ($\mathcal{F}$ being the so-called feature space) such that $\forall x, y \in \mathcal{X}$, $K(x, y) = \langle \varphi(x), \varphi(y) \rangle$. Note that this associated nonlinear mapping can be explicitly built (*e.g.* polynomial kernels) or not (*e.g.* Gaussian kernels). Thus, any algorithm which solely uses dot products can be cast by this kernel trick into a nonlinear procedure by implicitly mapping the original space $\mathcal{X}$ to the higher dimensional one $\mathcal{F}$.

However the kernel trick has some limitations. It cannot be applied to algorithms which imply to compute linear transformations of the form $x' = \sum_{i=1}^{m} \lambda_i x_i$. For example, in the k-means algorithm, new prototypes are computed as the centroid of associated data points, that is informally $x' = \frac{1}{m} \sum_{i=1}^{m} x_i$, or in Kohonen maps algorithms, a prototype $c$ has to be moved toward a new input $x$, that is $c \leftarrow c + \lambda(x - c)$ where $\lambda$ can be understood as a moving ratio. Performing linear transformations in the feature space in order to kernelize any vector quantization (VQ) algorithm is the problem addressed in this paper.

To do so, a dictionary method [2] is used so as to work directly in an approximation of the feature space $\mathcal{F}$, still without using $\varphi$ explicitly. Section 2 reviews some related works. Section 3 presents the dictionary method which is a kernel

ESANN'2009 proceedings, European Symposium on Artificial Neural Networks - Advances in Computational Intelligence and Learning. Bruges (Belgium), 22-24 April 2009, d-side publi., ISBN 2-930307-09-9.

sparsification procedure based on an approximate linear dependency argument. The Kernel-based Vector Quantization (KVQ) method is presented in Section 4. Notice that batch and online algorithms can be envisioned. Section 5 concludes and sketches future works.

## 2 Related Works

In this section some approaches for kernel clustering and vector quantization are briefly reviewed. Two types of algorithms will be distinguished: metric-based methods and feature space methods. Furthermore, the reader can refer to [3] for a survey on kernel (and spectral) methods for clustering and vector quantization.

A distance in the feature space can be computed using the kernel trick. For a specific kernel $K$ and the associated mapping $\varphi$, the distance between the images of two elements $x$ and $y$ of $\mathcal{X}$ can be computed using the bilinearity of the dot product and the kernel trick: $\|\varphi(x) - \varphi(y)\|^2 = K(x,x) - 2K(x,y) + K(y,y)$. Some approaches use this kernelized metric to directly cluster in the feature space, using algorithms which are expressed solely with dot products and distances. This approach is used in [4] to kernelize the k-means algorithm. It is applied in [5] to the Growing Neural Gas (GNG), however prototypes movements are done in the working space $\mathcal{X}$ (and thus the GNG is not fully kernelized). A k-means like approach using a stochastic optimization is introduced in [6] and extended in [7]. Our approach benefits from this metric view, however it applies to a broader class of algorithms.

Another approach is to directly work in the feature space for algorithms which implies linear transformations. State-of-the-art methods usually express a point in the feature space as a linear combination of *all* images of data points, that is $\mathbf{y} = \sum_{i=1}^{n} a_i \varphi(x_i)$ with $n$ being the size of the data set. An update rule for these linear combination's weights is then derived, depending on the algorithm at sight. Note that the nonlinear mapping $\varphi$ is never explicitly computed. Using this idea, the Self Organizing Map (SOM) is kernelized in [8] and [9], the Neural Gas (NG) algorithm in [10] and a fuzzy topographic clustering algorithm in [11]. This approach can be (more or less directly) applied to a larger class of algorithms than the previous one, as it allows computing linear transformations in the feature space. However it is computationally inefficient, as *all* data points are considered. The update rule for the linear combination weights has to be derived for each algorithm. Moreover, it does not work for online algorithms, for which data points are not known beforehand. The method proposed in this paper is close to the latter ones, as a feature vector is expressed as a linear combination of data points images. However, not all of them are necessary, and the set of data points to be used can be constructed online, thus this contribution overcomes the previous difficulties: it is more *generic* (there is no need to derive a specific rule for each algorithm), it can be applied to batch and *online* algorithms, and it is computationally *cheaper*, as a sparse representation of the feature space is maintained.

ESANN'2009 proceedings, European Symposium on Artificial Neural Networks - Advances in Computational Intelligence and Learning. Bruges (Belgium), 22-24 April 2009, d-side publi., ISBN 2-930307-09-9.

## 3 Dictionary Computation

As said in Section 1, the kernel trick corresponds to a dot product in a higher dimensional space, associated with a mapping $\varphi$. By observing that although $\mathcal{F}$ is a (very) higher dimensional space, $\varphi(\mathcal{X})$ can be a quite smaller embedding, the objective is to find a set of $p$ points in $\mathcal{X}$ such that $\varphi(\mathcal{X})$ is approximately embedded in Span $\{\varphi(\tilde{x}_1), \ldots, \varphi(\tilde{x}_p)\}$ and that $\{\varphi(\tilde{x}_1), \ldots, \varphi(\tilde{x}_p)\}$ are approximately linearly independent [2].

This procedure is iterative. Suppose that samples $x_1, x_2, \ldots$ are sequentially observed. At time $t$, a dictionary $\mathcal{D}_{t-1} = (\tilde{x}_j)_{j=1}^{p_{t-1}} \subset (x_j)_{j=1}^{t-1}$ containing $p_{t-1}$ elements is available where by construction feature vectors $\varphi(\tilde{x}_j)$ are approximately linearly independent in $\mathcal{F}$. Sample $x_t$ is then observed and is added to the dictionary if $\varphi(x_t)$ is (approximately) linearly independent on $\mathcal{D}_{t-1}$. To test this, weights $a = (a_1, \ldots, a_{p_{t-1}})^T$ have to be computed so as to verify $\| \sum_{j=1}^{p_{t-1}} a_j \varphi(\tilde{x}_j) - \varphi(x_t) \|^2 \leq \nu$, where $\nu$ is a predefined threshold determining the quality of the approximation (and consequently the sparsity of the dictionary). This can be solved by resolving $\delta_t = \min_{a \in \mathbb{R}^{p_{t-1}}} \| \sum_{j=1}^{p_{t-1}} a_j \varphi(\tilde{x}_j) - \varphi(x_t) \|^2$. If $\delta_t > \nu$, feature vectors are approximately independent and $x_t = \tilde{x}_{p_t}$ is added to the dictionary, otherwise not.

Using bilinearity of dot products, replacing them by kernels, and defining the $p_{t-1} \times p_{t-1}$ matrix $\tilde{K}_{t-1}$ and the $p_{t-1} \times 1$ vector $\tilde{k}_{t-1}(x)$ as $(\tilde{K}_{t-1})_{i,j} = K(\tilde{x}_i, \tilde{x}_j)$ and $(\tilde{k}_{t-1}(x))_i = K(x, \tilde{x}_i)$, the linear dependency test can be expressed in a matrix form: $\delta_t = \min_{a \in \mathbb{R}^{p_{t-1}}} \{a^T \tilde{K}_{t-1} a - 2a^T \tilde{k}_{t-1}(x_t) + K(x_t, x_t)\}$. This quadratic problem can be solved analytically and its solution is given by $a_t = \tilde{K}_{t-1}^{-1} \tilde{k}_{t-1}(x_t)$ and $\delta_t = K(x_t, x_t) - \tilde{k}_{t-1}(x_t)^T a_t$. If $\delta_t \leq \nu$, $\varphi(x_t)$ is approximately linearly dependent on $\mathcal{D}_{t-1}$ and can be written as $\varphi(x_t) = \sum_{i=1}^{p_{t-1}} a_i \varphi(\tilde{x}_i) + \varphi_t^{res} \approx \sum_{i=1}^{p_{t-1}} a_i \varphi(\tilde{x}_i)$, with $\|\varphi_t^{res}\| \leq \sqrt{\nu}$. Otherwise, if $\delta_t > \nu$, $x_t = \tilde{x}_{p_t}$ is added to the dictionary (approximate linear independence). See [2] for more details and figure 1 for an illustration. In the rest of this paper, the space embedded in $\mathcal{F}$ and spanned by the basis $\varphi(\mathcal{D})$ will be written $\tilde{\mathcal{F}}_{\mathcal{D}}$. Bold variables $\mathbf{x}$ will represent elements of the space spanned by the images of the dictionary elements, and classic variables $x$ will represent elements of the working space $\mathcal{X}$.

## 4 Kernel Vector Quantization

Suppose that a basis $\varphi(\mathcal{D}) = \{\varphi(\tilde{x}_1), \ldots, \varphi(\tilde{x}_p)\}$ is available. Recall that points $\tilde{x}_1, \ldots, \tilde{x}_p$ are explicitly known, but not their images $\varphi(\tilde{x}_1), \ldots, \varphi(\tilde{x}_p)$. Any feature vector $\varphi(x) \in \mathcal{F}$, with $x \in \mathcal{X}$, can be approximately expressed in this basis: there exists a vector $a^{\mathbf{x}} = (a_1^{\mathbf{x}}, \ldots, a_p^{\mathbf{x}})^T$ such that $\varphi(x) \approx \mathbf{x} = \sum_{i=1}^{p} a_i^{\mathbf{x}} \varphi(\tilde{x}_i)$. As seen in Section 3 the vector $a^{\mathbf{x}}$ can be easily computed. The principle of the approach presented here is to directly work in $\tilde{\mathcal{F}}_{\mathcal{D}}$, the image $\varphi(x)$ of a point $x \in \mathcal{X}$ being represented by its coefficients $a^{\mathbf{x}}$ in the basis, if necessary.

As we work in the space spanned by the basis $\varphi(\mathcal{D})$, any linear combination of vectors $\varphi(\tilde{x}_1), \ldots, \varphi(\tilde{x}_p)$ can be considered. For a specific $\mathbf{y} \in \tilde{\mathcal{F}}_{\mathcal{D}}$, which can be written $\mathbf{y} = \sum_{i=1}^{p} a_i^{\mathbf{y}} \varphi(\tilde{x}_i)$, it cannot be assessed that there exists a point

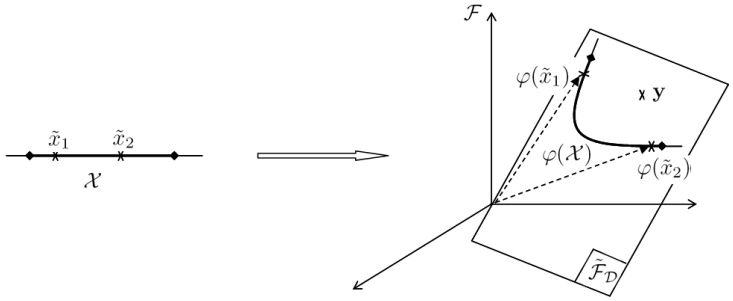


Fig. 1: The dictionary $\mathcal{D} = \{\tilde{x}_1, \tilde{x}_2\}$ is built from $\mathcal{X}$. The image of this dictionary $\varphi(\mathcal{D})$ is used to span $\tilde{\mathcal{F}}_\mathcal{D}$. The image $\varphi(\mathcal{X})$ of the working space $\mathcal{X}$ is approximately embedded in $\tilde{\mathcal{F}}_\mathcal{D}$, that is the distance between $\varphi(\mathcal{X})$ and $\tilde{\mathcal{F}}_\mathcal{D}$ is at the most $\sqrt{\nu}$. The point $\mathbf{y} \in \tilde{\mathcal{F}}_\mathcal{D}$ is a linear combination of $\varphi(\tilde{x}_1)$ and $\varphi(\tilde{x}_2)$, however it has no antecedent in $\mathcal{X}$.

$y \in \mathcal{X}$ such that $\varphi(y) = \mathbf{y}$, for example if $\mathbf{y}$ is an approximate linear combination of images of training points. See figure 1 for an illustration. The classical kernel trick applies to algorithms for which just dot products have to be computed, and it is here extended to algorithms for which linear transformations have to be computed too. The manner to do these two elementary operations in the posed framework is now shown.

### 4.1 Dot Product Computation

Suppose that the original algorithm requires computing a dot product between two points $x$ and $y$ of $\mathcal{X}$. By applying the kernel trick, one has to compute $\langle \varphi(x), \varphi(y) \rangle = K(x, y)$. If $x$ and $y$ are known, it is possible to directly compute $K(x, y)$. But recall that the proposed approach aims to directly work in the space spanned by $\varphi(\mathcal{D})$. Thus it is possible that only $\mathbf{x}$ or $\mathbf{y}$ in $\tilde{\mathcal{F}}_\mathcal{D}$ are known. We recall that there is no reason that there exists $x, y \in \mathcal{X}$ such that $\mathbf{x} = \varphi(x)$ or $\mathbf{y} = \varphi(y)$. Thus one can write $\mathbf{x} = \sum_{i=1}^{p} a_i^\mathbf{x} \varphi(\tilde{x}_i)$ and $\mathbf{y} = \sum_{i=1}^{p} a_i^\mathbf{y} \varphi(\tilde{x}_i)$. The dot product can still be computed: $\langle \mathbf{x}, \mathbf{y} \rangle = \langle \sum_{i=1}^{p} a_i^\mathbf{x} \varphi(\tilde{x}_i), \sum_{i=1}^{p} a_i^\mathbf{y} \varphi(\tilde{x}_i) \rangle = \sum_{i,j=1}^{p} a_i^\mathbf{x} a_j^\mathbf{y} K(\tilde{x}_i, \tilde{x}_j) = (a^\mathbf{x})^T \tilde{K} a^\mathbf{y}$, $\tilde{K}$ being defined in section 3. If $\mathbf{x}$ lies in the span of $\varphi(\mathcal{D})$ and $y$ in the working space then the dot product between $\mathbf{x}$ and $\varphi(y)$ can be computed as $\langle \mathbf{x}, \varphi(y) \rangle = \langle \sum_{i=1}^{p} a_i^\mathbf{x} \varphi(\tilde{x}_i), \varphi(y) \rangle = (a^\mathbf{x})^T \tilde{k}(y)$, $\tilde{k}$ being defined in section 3. Recall that computing a distance yields to the evaluation of a dot product, thus $\|\mathbf{x} - \mathbf{y}\|^2 = (a^\mathbf{x} - a^\mathbf{y})^T \tilde{K} (a^\mathbf{x} - a^\mathbf{y})$ and $\|\mathbf{x} - \varphi(y)\|^2 = (a^\mathbf{x})^T \tilde{K} a^\mathbf{x} - 2(a^\mathbf{x})^T \tilde{k}(y) + K(y, y)$.

The proposed approach is thus not restrictive and still allows computing distances and dot products, as the kernel trick standard approach. However a point of $\tilde{\mathcal{F}}_\mathcal{D}$ has not necessarily an antecedent and this can induce more computation than just applying a kernel function.

ESANN'2009 proceedings, European Symposium on Artificial Neural Networks - Advances in Computational Intelligence and Learning. Bruges (Belgium), 22-24 April 2009, d-side publi., ISBN 2-930307-09-9.

**Algorithm 1**: Kernel Vector Quantization

**Compute** dictionary $\mathcal{D}$ and matrixes $\tilde{K}$ and $\tilde{K}^{-1}$, **from** the given samples or from a compact set, **in** a preprocessing step (batch) or online ;

**Replace** any training point $x$ by the associated vector $a^{\mathbf{x}} = \tilde{K}^{-1}\tilde{k}(x)$ of $\mathbf{x} = \varphi(x)$ expressed in the basis $\varphi(\mathcal{D})$ ;

**Replace** any dot product $\langle x, y\rangle$ by the dot product $(a^{\mathbf{x}})^T \tilde{K} a^{\mathbf{y}}$. Use directly $K$ or $\tilde{k}$ if possible (see section 4.1);

**Replace** any distance $\|x - y\|^2$ by the distance $(a^{\mathbf{x}} - a^{\mathbf{y}})^T \tilde{K}(a^{\mathbf{x}} - a^{\mathbf{y}})$. Use directly $K$ or $\tilde{k}$ if possible (see section 4.1);

**Replace** any linear transformation $y = (x_1, \ldots, x_m)(\lambda_1, \ldots, \lambda_m)^T$ by the corresponding linear transformation $a^{\mathbf{y}} = (a^{\mathbf{x}_1}, \ldots, a^{\mathbf{x}_m})(\lambda_1, \ldots, \lambda_m)^T$

### 4.2 Linear Transformation Computation

Suppose that the original algorithm requires computing a linear transformation $x' = \sum_{i=1}^{m} \lambda_i x_i$. In order to be consistent with the kernel trick, kernelizing this kind of algorithm implies computing a point $\mathbf{x}' \in \mathcal{F}$ such that $\mathbf{x}' = \sum_{i=1}^{m} \lambda_i \mathbf{x}_i$ with $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \tilde{\mathcal{F}}_{\mathcal{D}}$ and using the same $(\lambda_i)_{i=1}^m$. There is a simple way to do so directly in the approximated feature space.

From the fact that no more than linear combinations are considered, each vector $\mathbf{x}_i$ can be expressed as $\mathbf{x}_i = \sum_{j=1}^{p} a_j^{\mathbf{x}_i} \varphi(\tilde{x}_j)$. It is thus easy to express $\mathbf{x}'$ in the same basis: $\mathbf{x}' = \sum_{i=1}^{m} \lambda_i (\sum_{j=1}^{p} a_j^{\mathbf{x}_i} \varphi(\tilde{x}_j)) = \sum_{j=1}^{p} (\sum_{i=1}^{m} \lambda_i a_j^{\mathbf{x}_i}) \varphi(\tilde{x}_j)$. That is $\mathbf{x}'$ is associated with the coordinates $a^{\mathbf{x}'} = (\sum_{i=1}^{m} \lambda_i a_1^{\mathbf{x}_i}, \ldots, \sum_{i=1}^{m} \lambda_i a_p^{\mathbf{x}_i})^T = A\lambda$ with $A$ being the $p \times m$ matrix $(a^{\mathbf{x}_1}, \ldots, a^{\mathbf{x}_m})$ of coordinate column vectors $a^{\mathbf{x}_i}$ and $\lambda$ being the $m \times 1$ vector $(\lambda_1, \ldots, \lambda_m)^T$.

Linear transformations are therefore made possible through the use of the approximate feature space. With the proposed KVQ, any VQ algorithm (which solely implies computing dot products and linear transformations) can be *straightforwardly* kernelized. Notice that the KVQ can be applied to batch and online algorithms. For batch algorithms, the dictionary can be computed from the data set or from a compact set $X$ embedding the working space $\mathcal{X}$. In the case of online algorithms, the dictionary can be computed online as training sample are observed or in a pre-processing step from a compact set embedding the working space. The KVQ is summarized in algorithm 1.

## 5 Conclusion

We have proposed a novel generic approach which allows extending the field of algorithms that can benefit from the kernel trick to any algorithm which only requires computing dot products and linear transformations. Moreover, the KVQ can be applied to batch and online algorithms. Although the obtained results are approximated, this can be seen as an efficient extension of the clas-

sical kernel trick which can provide itself useful for kernel vector quantization algorithms. Compared to papers reviewed in section 2, the KVQ allows *handling linear transformations* (compared to metric-based approaches), it is *generic* (it can be systematically applied to any VQ algorithm, contrary to other feature space approaches), it can be applied to *batch and online* algorithms and it is *computationally cheaper* than other feature space approaches, as a sparse representation of the feature space is maintained. Moreover the trade-off between sparsity and accuracy can be controlled through the choice of the sparsity factor. For now, some VQ algorithms have been kernelized [12], and especially GNG-T, a new neural gas algorithm [13]. Some results are provided on the web page. Work is ongoing to use kernelized VQ algorithms as a preprocessing step for Support Vector Machines (SVM) and to compare distortions in working and feature spaces.

## Acknowledgement

The authors would like to acknowledge Hervé Frezza-Buet for developing the KVQ `C++` library available online [12].

## References

[1] V. N. Vapnik. *Statisical Learning Theory*. John Wiley & Sons, Inc., 1998.

[2] Y. Engel, S. Mannor, and R. Meir. The Kernel Recursive Least Squares Algorithm. *IEEE Transactions on Signal Processing*, 52(8):2275–2285, 2004.

[3] M. Filippone, F. Camastra, F. Masulli, and S. Rovetta. A survey of kernel and spectral methods for clustering. *Pattern Recogn.*, 41(1):176–190, 2008.

[4] S. Vishwanathan and Narasimha M. Murty. Kernel Enabled K-Means Algorithm. Technical report, National ICT Australia, Statistical Machine Learning Program, 2002.

[5] L. d'Amato, J. A. Moreno, and R. Mujica. Reducing the Complexity of Kernel Machines with Neural Growing Gas in Feature Space. In *IBERAMIA*, pages 799–808, 2004.

[6] M. Girolami. Mercer kernel-based clustering in feature space. *IEEE Transactions on Neural Networks*, 13(3):780–784, May 2002.

[7] D. S. Satish and C. C. Sekhar. Kernel based clustering and vector quantization for speech recognition. In *14th IEEE Signal Processing Society Workshop*, pages 315–324, 2004.

[8] R. Inokuchi and S. Miyamoto. LVQ Clustering and SOM Using a Kernel Function. *Journal of Japan Society for Fuzzy Theory and Intelligent Informatics*, 17(1):88–94, 2005.

[9] D. MacDonald and C. Fyfe. A new kernel clustering algorithm. In *International Conference on Knowledge-Based Intelligent Engineering Systems and Allied Technologies*, volume 1, pages 317–320, 2000.

[10] A. K. Qin, , and P. N. Suganthan. Kernel Neural Gas Algorithms with Application to Cluster Analysis. In *17th International Conference on Pattern Recognition*, volume 4, pages 617–620, Washington, DC, USA, 2004. IEEE Computer Society.

[11] T. Graepel and K. Obermayer. Fuzzy Topographic Kernel Clustering. In W. Brauer, editor, *5th GI Workshop Fuzzy Neuro Systems 98*, pages 90–97, 1998.

[12] H. Frezza-Buet. KVQ `C++` library. http://ims.metz.supelec.fr/spip.php?article87, 2008.

[13] H. Frezza-Buet. Following non-stationary distributions by controlling the vector quantization accuracy of a growing neural gas network. *Neurocomputing*, 71(7-9):1191–1202, 2008.