Classification approach based on non-negative least squares

Yifeng Li*, Alioune Ngom

School of Computer Science, 5115 Lambton Tower, University of Windsor, 401 Sunset Avenue, Windsor, Ontario, Canada N9B 3P4

A R T I C L E   I N F O

Article history:
Received 21 June 2012
Received in revised form 22 November 2012
Accepted 6 February 2013
Communicated by G. Thimm

Keywords:
Sparse representation
Non-negative matrix factorization
Classification
Kernel approach
Missing values

A B S T R A C T

A non-negative least squares classifier is proposed in this paper for classifying under-complete data. The idea is that unknown samples can be approximated by sparse non-negative linear combinations of few training samples. Based on sparse coefficient vectors representing the training data, a sparse interpreter can then be used to predict the class label. We have devised new sparse methods which can learn data containing missing value, which can be trained on over-complete data, and which also apply to tensor data and to multi-class data. Permutation test shows that our approach requires a small number of training samples to obtain significant accuracy. Statistical comparisons on various data shows that our methods perform as well as support vector machines while being faster. Our approach is very robust to missing values and noise. We also show that with appropriate kernel functions, our methods perform very well on three-dimensional tensor data and run fairly fast.

© 2013 Elsevier B.V. All rights reserved.

1. Introduction

Various large-scale biological data have been generated within the last decade. Typical such high-throughput data are microarray data [1] and mass spectrometry data [2]. Their common issue is that they contains very few number of samples compared to the large features; that is, they are under-complete data as opposed to over-complete data where the number of samples is (much) larger than the number of features. In medical research, the expression levels of genes with respect to biological samples can now be monitored synchronically over a series of time-points. This corresponds to a three-dimensional (3D) dataset, termed gene-sample-time (GST) microarray data [1,3], which can be viewed as a collection of gene-sample data over a series of time-points, or a collection of gene-time data across some samples. These are examples of 3-order tensor data [4]. An order-\textit{n} tensor data is a vector having \textit{n} directions, for example, a vector, a matrix, and a cube are tensor of orders 1, 2, and 3, respectively. Analyzing (i.e., classifying or clustering) tensor data of order three or more is more challenging due to the extra number of dimensions. High through-put data provide much richer information for biologists, however they are quite challenging to mine and analyze due to their low sample size and their large number of (redundant or irrelevant) features. The issues derived from analyzing the undercomplete data are called “the curse of dimensionality”. Many classifiers confront with problems when classifying such data. Some classifiers do not theoretically work at all, for example, Gaussian Bayesian classifier [5] might fail to calculate the inverse of singular covariance matrix. Some are practically expensive to execute, such as decision tree [6] and neural networks [7]. If an inductive model is not consistent, then it would result in overfitting, that is poor capability of generalization [8].

One solution to the curse of dimensionality is dimension reduction which includes feature extraction and feature selection. 

Principle component analysis (PCA) [9] and independent component analysis (ICA) [10] are among the most popular dimension reduction techniques. Non-negative matrix factorization (NMF) has been intensively studied since the work of Lee and Seung [11]. For example, NMF is studied as feature extraction and feature selection methods in Li and Ngom [12]. Alternatively, sophisticated models have been directly applied to classifying high-dimensional data, of which the most famous family being the basis-expanded linear models which will be introduced in Section 3. Noise is another issue which cannot be avoided completely and can result in overfitting issues if a model is not robust. Sparse representation (SR) [13,14] has been proven very robust to noise and redundancy. It has been used to classify under-complete and over-complete data. The sparse representation classification (SRC) approach has been reported to be robust to random pixel corruption [15] in image processing. For instance, on a facial dataset where even the test images are 70% corrupted, SRC surprisingly obtained an accuracy of over 90.7%. Besides, sparse representation has been also applied successfully as a de-noising method [16]. Please see Section 3 for details. High-throughput...
data often contain missing values and many classifiers are not devised for such to handle incomplete data well. Bayesian network based prediction methods become extremely slow when dealing with missing values since expectation maximization (EM) strategy is used [17,18] to deal with the missing values. Decision trees can deal with missing values, but may fail beyond a certain percentage of missing values in the data. In this paper, we propose a novel non-parametric classification method which directly classifies the original under-complete data. The sparse codes of new samples are generated via non-negative constraints instead of the $l_1$-norm minimization. Our approach can efficiently predict the class labels of a set of unknown samples in parallel. We propose a robust weighting strategy to handle incomplete data (i.e., data containing missing values). Our approach can naturally work on multi-class data. Our optimization algorithms use only the inner products of samples and therefore are dimension-free. Applying this property, kernel optimization algorithms use only the inner products of samples values). Our approach can naturally work on multi-class data. Our strategy to handle incomplete data (i.e., data containing missing values) is to deal with partial observations since $\%$ of missing values in the data.

2. Notations

Hereafter we shall use the notations as listed below, unless otherwise noted.

(1) A scalar is denoted by an italic lowercase or upper case letter, for example, $a$ and $A$.

(2) A matrix is denoted by a bold capital letter, for example, $Y$. A bold lowercase letter with a subscript denotes a column vector of a matrix. The colon notation, "::", in the subscript of a matrix denotes taking all elements in the corresponding order. For example, $y_i = y_{i,:}$ is the $i$-th column vector in matrix $Y$. The $j$-th row of $Y$ is $Y_{:,j}$. An italic lowercase letter with two subscripts represents a scalar element of the corresponding matrix. For example, $y_{ij}$ is the $(ij)$-th scalar element of matrix $Y$.

(3) A tensor is denoted by a boldface Euler script [4], for example, $X$.

(4) $X_{ij}$ denotes the matrix obtained through the mode-$n$ matricization of the tensor $X$ [4]. Columns of $X_{ij}$ are the mode-$n$ fibers of tensor $X$. A mode-$n$ fiber is a vector defined through fixing every index but the $n$-th index. This is the extension of matrix row and column in tensor algebra. $X_{ij}$ therefore denotes the matrix of size $I \times J$, unfolded in mode-1 of $X \in R^{I \times J \times K}$; that is, $X_{ij} = [X_{i1j}, X_{i2j}, \ldots, X_{ijn}]$.

(5) The mode-$n$ product of a tensor $X$ and a matrix $A$ [4], written as $X \times_n A$, is; $(X \times_n A)_i = \sum_{j=1}^d (x_{ij} \cdot a_j)$, where $X \in R^{I \times J \times K}$ and $A \in R^{J \times K}$. This results in a tensor $Y \in R^{I \times d}$.

(6) A set is denoted by an uppercase calligraphic letter, for example, $S$. If the set elements are also sets, bold font is used, for example, $S = \{\{1,2\},\{3,4\}\}$.

(7) A set containing finite number of continuous integers is defined by colon notation. For example, $N = \{1,2,\ldots,m\}$.

(8) A matrix can be indexed by a set. For example, $Y_{\mathcal{C}} = Y_{\mathcal{C}}$ denotes the columns of $Y$ indexed by set $\mathcal{C}$ and $Y_{\mathcal{R}}$, denotes the rows of $Y$ indexed by $\mathcal{R}$.

3. Related works

In this section, we survey existing approaches addressing part of the issues mentioned in Section 1. First, we shall introduce NMF and the classification approaches devised for under-complete data. We then review sparse representation methods which are robust to noise. Finally, we compared three strategies for handling missing values.

3.1. Non-negative matrix factorization for dimension reduction

Non-negative matrix factorization (NMF) has been intensively studied since the work of of Lee and Seung [11]. NMF decomposes a non-negative matrix $A$ into two low-rank non-negative factors: 

$$A_{m \times n} = W_{m \times k H_{k \times n}}$$

where $k \leq \min(m,n)$. In the Euclidean space, its optimization can be formulated as

$$\min_{W,H} \frac{1}{2} \|A - WH\|^2_2 \quad \text{s.t.} \quad W, H \geq 0. \quad (1)$$

In machine learning, if each column of $A$ is a training sample, then each training sample is approximated by a non-negative linear combination of the columns of $W$, that is, $a_i \approx \sum_{j=1}^{k} w_{ij} = W_{ij}$. Each column of $W$ is called a basis vector or metasample. Thus $W$ is called the basis matrix and $H$ is the coefficient matrix. Semi-NMF [19] allows negative values in $A$ and drops the non-negative constraints on $W$. NMF has been applied as feature extraction method as well as a clustering approach. The key difference between NMF and techniques like PCA and ICA is that, its basis vectors are non-orthogonal and its coefficient matrix is non-negative and usually sparse. The largest coefficient helps us to identify the basis vector (or cluster prototype) most related to a sample.

3.2. Basis-expanded linear model and kernel trick

The general linear model for classification can be formulated as $f(x) = \text{sign}(w^T\phi(x) + b)$ where $\phi(*)$ is a mapping function and $b$ is a bias. Given $n$ training samples $D = R^{m \times n}$ and the corresponding class information $c$, its risk minimization can be expressed as

$$\min_w r(w) + C \sum_j \phi(z_j)$$

where $r(w)$ is the regularization term, $\phi(z_j)$ is a loss function, and $C$ is the trade-off parameter. The first term is to increase the generalization of the model, whereas the second term is to decrease the empirical classification error. Most linear models can be kernelized as long as the optimization and prediction use only the inner products between the samples. The kernel trick can make the optimization dimension-free. An appropriate kernel can linearize complex patterns. When $r(w)$ is the $L_2$-norm and the loss function is the hinge loss, then this is the well-known support vector machine (SVM) [20]. It is well-known that SVM works well on both under-complete and over-complete data given proper kernels. One of the potential weaknesses of linear models is that they reduce the multi-class problem into many binary-class ones, which becomes inefficient as the number of class increases.

3.3. Independent modelling for multi-class data

Soft independent modelling of class analogies (SIMCA) [21] is a multi-class method that measures the (dis)similarity between a...
new sample to each class independently and assigns the new sample to the closest class. A popular dissimilarity measure is the regression residual. Unlike SIMCA which uses PCA to find the basis vectors of each class, linear regression classification (LRC) [22] directly uses all the training samples within a class to compute the ordinary-least-squares regression residual. LRC only works under the condition that each classes must be under-complete. Moreover, due to independence, it may not work well in the situation that the class distributions are entangled.

3.4. Sparse representation

Sparse representation [13] is a principle of parsimony which states that a new signal is approximated by a sparse superposition of basis functions. Suppose \( b \) is a \( m \)-dimensional signal, sparse representation can be formulated as \( b \approx \sum_{i=1}^{k} a_i x_i = Ax \), where \( A = [a_1, \ldots, a_k] \) is called a dictionary and \( a_i \) is a dictionary atom (basis function or basis vector). Sparse representation includes sparse coding and dictionary learning. Given a new signal \( b \) and the dictionary \( A \), learning the sparse coefficient \( x \) is a procedure of sparse coding. Given training data \( B \), learning the dictionary \( A \) from data is called dictionary learning. If \( m < k \), \( A \) is an over-complete dictionary. This means that \( b = Ax \) is under-determined and may have infinite number of solutions. The optimal sparse coefficient vector \( x \) is the optimal solution to the problem,

\[
\min \|x\|_0 \quad \text{s.t.} \quad b = Ax.
\]

where \( x^0 = \sum_{i=1}^{n} \delta(x_i) \) and \( \delta(x) = 1 \) if \( x \neq 0 \), and 0 otherwise. The constraint can be relaxed to \( \|b - Ax\|_2^2 \leq \epsilon \), where \( \epsilon \) is a small positive tolerance value. Unfortunately, this optimization is NP-hard [23]. Basis pursuit (BP) [24] is one of the heuristic approximation methods in which \( l_1 \)-norm minimization is performed as

\[
\min_x \|x\|_1 \quad \text{s.t.} \quad b = Ax.
\]

or the version allowing small relaxation

\[
\min_x \|x\|_1 \quad \text{s.t.} \quad \|b - Ax\|_2^2 \leq \epsilon.
\]  

An example of solvers to this problem is \( l_1 \)-magic [25].

If \( m > k \), then \( A \) is under-complete and there is no solution to the over-determined \( b = Ax \). The optimization task reduces then to finding a sparsest coefficient while minimizing the error \( \|b - Ax\|_2^2 \). The \( l_1 \)-regularized approximation of this problem is transformed into a linear combination of the objectives,

\[
\min_x \frac{1}{2} \|b - Ax\|_2^2 + \lambda \|x\|_1.
\]

Sparse representation has been applied in some fields such as computer vision [26,27], machine learning [28], and bioinformatics [29,30]. Sparse representation classification (SRC1), we added the numerical postfix to indicate that there exists other SRC approaches) is proposed for face recognition [15].

SRC1 constructs over-complete dictionary from the training samples after dimension reduction, then applies \( l_1 \)-norm minimization, as in Eq. (2), to pursue sparse coefficient vectors. It has been shown that SRC1 has state-of-the-art classification performance and is robust to noise. One of the benefits of sparsity is that it encodes a good semantic and discriminative information which facilitates interpretation and classification. SRC1 is a multi-class method. The way of pooling all training samples from all classes into a dictionary may works better than the independent modeling in the case of cluttered distributions. However, in its application to classifying high-dimensional data, some dimension reduction techniques (for example, downsampling) has to be employed to make sure the dictionary is over-complete.

 SRC2 based on Eq. (3) has been applied to classify high-dimensional gene expression data by Hang and Wu [29]. SRC2 works on under-complete dictionary. SRC2 can be solved by 11_1s [31]. One of the drawback of this algorithm is that the running time grows dramatically as the number of dictionary atoms increases [29]. Also, the value of parameter \( \lambda \) depends on specific tasks and hence has to be optimized. This is an expensive procedure. Moreover, given a set of new samples, the queries are sequential, which also aggravates the computational burden.

Dictionary learning [13] is an interesting topic in sparse representation to extract hidden patterns from data. The benefits of dictionary learning are that (1) the number of dictionary elements learned can be much smaller than the number of training samples, which expedites the sparse coding, and (2) prediction performance might be improved on the learned dictionary. Aiming at reducing the running time while increasing the classification performance of SRC2, Zheng et al. [30] proposed a metasample-based sparse representation classification (MSRC) approach. A metasample is a basis vector. A set of metasamples is extracted for each class. Metasamples can be extracted by NMF or SVD. These sets of metasamples of all classes are concatenated as a holistic under-complete dictionary. After learning the dictionary \( A \), for a new sample, MSRC optimizes Eq. (3) to obtain the corresponding sparse coefficient vector. This sample is assigned to a class using the nearest-subspace rule. Besides selecting the trade-off between the least squares error and the \( l_1 \)-norm regularization in the objective, the model selection introduced in the dictionary learning step is also time-consuming and degrades dramatically as the number of classes increases.

3.5. Handling incomplete data

There are three strategies to handle the issue of missing values. First, we can remove the features or samples with missing values. However, the main drawback is that the already small sample size becomes smaller and we may run the risk of having not enough data for learning a satisfactory and consistent model. The second strategy is to impute (that is, to fill-in) the missing values in the data [32]. Missing values can either be imputed by a constant value (the most common value is 0 which corresponds to 0-imputation) or features averages, or alternatively they can be estimated by some statistical, or supervised or unsupervised learning methods. Expectation maximization (EM) and \( k \)-nearest neighbor (k-NN) are two of the most well-known estimation methods [17,32]. This strategy essentially completes an incomplete dataset, and hence, avoids deleting features or samples. The time complexity of making data complete before learning through this way is usually much lower than that of hybridizing EM and a learning method. Nevertheless, the issue of imputation is that false data can be introduced. The situation can even be worse if the missing data do not actually exist. Moreover, the larger the missing rate is, the more difficult for an estimation method to execute. The third strategy is that the present values are only allowed to be used in learning and prediction.

3.6. Single-left-hand-side non-negative least squares

We will apply an active-set algorithm to solve a variant of the multiple-left-hand-side non-negative least squares (MLHS-NLNS) in Section 4.3. For the sake of understanding, we first describe the active-set algorithm for the single-left-hand-side non-negative least squares (SLHS-NLNS, Eq. (5)). Active-set algorithms solving NNLS problems have been developing since 1970s. The standard algorithm was proposed by Lawson and Hanson [33] to solve the SLHS-NLNS problem which is shown in Algorithm 1. For narrative convenience, we call this algorithm LH-NLNS algorithm.
Algorithm 1. LH-NNLS optimization algorithm.

Input: matrix $A_{m \times n}$, vector $b_{m \times 1}$
Output: vector $x = \arg \min_{x} \frac{1}{2} \|b - Ax\|^2_2 \text{ s.t. } x \succeq 0$

$x \leftarrow 0$; 
$\mathcal{R} \leftarrow \{i | i = 1 : n\}$; initialize the active set 
$\mathcal{P} \leftarrow \emptyset$; initialize the passive set 
\[ \mu \leftarrow A^T b - A^T A x; \] 
the lagrange multiplier 
\[ \epsilon \leftarrow (0, 1]; \] 
small positive tolerance value 
while $\mathcal{R} \neq 0$ and $\max_{i \in \mathcal{R}} (\mu_i) > \epsilon$
do 
\[ j \leftarrow \arg \max_{i \in \mathcal{R}} \mu_i; \] 
index with maximal positive multiplier 
$\mathcal{P} \leftarrow \mathcal{P} \cup \{j\}$; 
$\mathcal{R} \leftarrow \mathcal{R} \setminus \{j\}$; 
$\theta_{ij} \leftarrow (\lambda_j A_{ij} + \mu_j) A^T_{ix}$; 
$\theta_{\overline{x}j} \leftarrow 0$; 
while $\min_{\mathcal{P}} \leq 0$
do 
\[ x \leftarrow \min_{i \in \mathcal{P}} \left( x_i + \frac{x_j}{\lambda_j} \right); \] 
$\mathcal{K} \leftarrow \arg \min_{i \in \mathcal{P}} \min_{x_i \leq 0} \frac{x_j}{\lambda_j} = \text{many indices for } x$ 
$x \leftarrow x + \alpha (I - \xi); \] 
$\mathcal{P} \leftarrow \mathcal{P} \cup \mathcal{K}$; 
$\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{K}$; 
$\theta_{ij} \leftarrow (\lambda_j A_{ij} + \mu_j) A^T_{ix}$; 
$\theta_{\overline{x}j} \leftarrow 0$; 
end while 
$x \leftarrow t$; 
$\mu \leftarrow A^T b - A^T A x$; 
end while 

This algorithm iteratively searches the true active set and the passive set. The true active set and the passive set are, respectively, the set of variables whose values lie on the border of a feasible region and the set of variables whose values are inside the feasible region. LH-NNLS is a rigorous algorithm which can converge to optimal solution. The convergence is proved in the following. The Lagrangian function for the LH-NNLS problem in Algorithm 1 is $L(x, \mu) = 1/2 \|b - Ax\|^2_2 + \mu^T x$. The corresponding Karush–Kuhn–Tucker (KKT) conditions are
\begin{align*}
A^T (Ax - b) + \mu &= 0 \quad , \\
\mu \ast x &= 0 \quad , \\
\mu \leq 0, \ x \succeq 0
\end{align*}
where $\ast$ is the component-wise multiplication operator. When the algorithm terminates, $\mu_{\overline{x}} < 0$ and $\mu_x = 0$, where $\mathcal{R}$ and $\mathcal{P}$ are the true active set and passive set, respectively. That is, 
\begin{align*}
\mu_{\overline{x}} &= A^T_{\overline{x}} b_{\overline{x}} - A^T_{\overline{x}} A_{\overline{x}} x_{\overline{x}} < 0, \\
\mu_x &= A^T_x b_x - A^T_x A_x x_x = 0.
\end{align*}
This means that the KKT conditions in Eq. (4) are satisfied. The Hessian of $L(x, \mu)$ is $A^T A$ which is positive definite. Also because $\mu_{\overline{x}} < 0$, the solution returned by LH-NNLS is a strict local minimum. Since the objective function and the feasible region are convex, this solution is thus a global minimum [34]. In the optimal solution, the values corresponding to the active set are set to 0 and the remaining values are the optimal solution to the unconstrained least squares subproblem.

4. Proposed method

In this section, we first propose our classification approach based on non-negative sparse coding. We then give a numerical example to demonstrate how our method works. Next, we address the optimization of our method. Thereafter, we investigate the relation between sparsity and non-negativity constraints. We also propose a weighting strategy for handling missing values. Finally, the kernel version of our approach and its application for tensor data are described.

4.1. Non-negative least-squares classification approach

We propose non-negative-least-squares (NNLS) classification approach in which any new sample with unknown class label is approximated by a non-negative linear combination of the training samples. The combination coefficients are sparse due to the non-negativity constraints. Suppose $b_{m \times 1}$ is a new sample with unknown label and $A_{m \times n}, m > n$, is the training set with $n$ samples in columns and $m$ features in rows. We have $b \approx \sum_{i=1}^{m} a_i x_i = A x$, where the coefficient vector $x \succeq 0$. Because $A$ is under-complete, $b = A x$ is usually overdetermined. This means there does not exist a non-negative solution $x$. Therefore, obtaining $x$ is a single-left-hand-side NNLS (SLHS-NNLS) problem formulated as
\begin{equation}
\min_{x} \frac{1}{2} \|b - A x\|^2_2 \text{ s.t. } x \succeq 0.
\end{equation}
If there are $p \geq 2$ new samples, there is no need to solve the NNLS problem one by one as in Eq. (5). Instead, we can solve the NNLS problems in batch. The approximation becomes $B = A X$, where $B$ is of size $m \times p$, $X$ is of size $n \times p$, and $X \succeq 0$. Each column of $B$ is a new sample. The j-th column of $X$, $x_j$, is the coefficient vector corresponding to the j-th new sample, $b_j$, that is $b_j \approx A x_j$. And the optimization task becomes the multiple-left-hand-side NNLS (MLHS-NNLS) problem,
\begin{equation}
\min_{X} \frac{1}{2} \sum_{i=1}^{p} \|b_i - A X_i\|^2_2 \text{ s.t. } X \succeq 0.
\end{equation}
This problem is equivalent to
\begin{equation}
\min_{X} \frac{1}{2} \sum_{i=1}^{p} \|b_i - A x_i\|^2_2 \text{ s.t. } X \succeq 0.
\end{equation}

After obtaining the sparse coefficients, we need to employ a sparse interpreter to predict the class of unknown samples. We shall give two interpreters (or rules), in the following.

The rule that assigns the class label of the training sample with the largest coefficient to the new sample is called the MAX rule. This rule is inspired by the rule used in NMF clustering method. The relation between our NNLS method using the MAX rule and 1-NN is that the MAX rule is actually 1-NN in the column space of $A$, where the representation of the i-th training sample is a vector with the i-th component to be 1 and the rest 0. The representation of a new sample in this space is the corresponding sparse coefficient vector. Identifying the largest coefficient corresponds to finding the closest training sample in the column space of $A$.

The sparsity may decrease as the noise increases in the data. There may not exist dominantly large coefficients for very noisy data. In such case, incorrect decision may be made by the MAX rule. The nearest-subspace (NS) rule is proposed by Wright et al. [15] to interpret the sparse coding. This rule takes the advantage of the property of discrimination of sparse coefficients and is more robust to noise than the MAX rule. We can also include this rule in our NNLS approach as the sparse interpreter. Suppose there are C classes with labels $l_1, \ldots, l_C$, for a given new sample $b$, after obtaining its non-negative coefficient vector $x$, the regression residual corresponding to the k-th class is computed as
\begin{equation}
r_k(b) = \frac{1}{2} \|b - A \delta_{k}(x)\|^2_2,
\end{equation}
where $\delta_k(x) = x \mathbb{1}_{C_k}$, $C_k = \{i | l_i = k\}$, and $\mathbb{1}$ is the indicator function of the i-th class.
where \( \delta_k(x) : \mathbb{R}^n \rightarrow \mathbb{R}^n \) returns the coefficients for class \( l_k \). Its \( j \)-th element is defined by
\[
(\delta_k(x))_j = \begin{cases} 
  x_j & \text{if } a_j \text{ in class } l_k, \\
  0 & \text{otherwise}.
\end{cases}
\]

Finally, class label \( l_k \) is assigned to \( b \), where
\[
K = \arg \min_{1 \leq k \leq C} r_k(b).
\]

(8)

Our NNLS based classification simply involves two steps that are listed in Algorithm 2. First of all, we need a fast optimization algorithm (see Section 4.3) to obtain the non-negative coefficient matrix \( X \). Then the class label of each new sample is predicted by a sparse interpreter as explained above. For narrative convenience, we list the NNLS approach with MAX rule by NNLS-MAX and that with the nearest-subspace rule by NNLS-NS.

**Algorithm 2.** Non-negative least squares classifier.

**Input:** \( A_{m \times n} \): training set including \( m \) features and \( n \) samples; \( c \): class labels of the training samples, \( b_{m \times p}: p \) new samples

**Output:** \( p \): predicted class labels of the \( p \) new samples

1. (1) sparse coding: solve the NNLS optimization problem in Eq. (6);
2. (2) sparse interpreter: use MAX or NS rule to predict the class labels of the new samples: \( p = \text{MAX}(X) \) or \( p = \text{NS}(X) \);

In Algorithm 2, function MAX returns a vector \( p \) containing the predicted class labels of the new samples from \( B \). The \( i \)-th component of \( p \) is defined as \( p_i = c_{i_k} \), if \( X_{i_k} \) is the largest value in the \( i \)-th column of \( X \), that is \( K = \arg \max_{1 \leq i \leq n} (X_{i_k}) \). Suppose there are \( C \) classes with labels \( l_1, \ldots, l_C \). The \( i \)-th component of \( p \) returned by function NS is defined as \( p_{i_k} \), where \( K = \arg \min_{1 \leq i \leq n} (r_{i_k}(b)) \). Our NNLS classifier is an example of instance-based learning method and it is well-known that instance-based learning algorithms perform well for complicated distributions [7]. NNLS-NS can be also viewed as a locally weighted linear regression approach [7,35] except that (1) NNLS-NS finds the approximation coefficients through non-negative least squares, while locally weighted regression usually involves a distance-based weighting and (2) NNLS-NS finds the local training instances adaptively with respect to the contribution to reduce the regression residual further, whereas locally weighted regression employs distance-based \( k \)-NN to find local instances surrounding the new query. Furthermore, our NNLS-MAX classifier is a locally weighted classification method, because it actually uses a decision function over the locally weighted regression as defined by \( g(\arg \min_{0 \leq i \leq 1/2} |b - Ax|_2) \), where the decision function is \( g(x) = \text{MAX}(x) \). Finally, we should note that NNLS-MAX is not equivalent to 1-NN in the input space.

### 4.2. Numerical example

Now let us have three simulation examples to demonstrate how this method works for both non-noisy and noisy data. First, suppose \( A \) contains 6 samples with 8 features (see Eq. (9)). And the first 3 samples belong to class 1 and the last 3 are from class 2. Also suppose there is no intergroup variation and noise. Suppose we have 4 new samples in \( B \). We can obtain its coefficient matrix \( X \) by a NNLS algorithm. The largest values in the columns of \( X \) exactly indicate that the first 2 new samples have the same class labels with the first sample in \( A \) and the last 2 new samples have the same class labels with the forth sample in \( A \). Second, if we add a little portion of Gaussian noise to \( A \) (through \( A = A + 0.1 \times \text{randn}(8,6) \), where \( \text{randn}(8,6) \) returns a \( 8 \times 6 \) matrix, each element of which is generated by standard Gaussian generator) and \( B \) (through \( B = B + 0.1 \times \text{randn}(8,4) \)), we would obtain Eq. (10). From the dominantly largest values in the columns of \( X \), we can predict that the first two new samples fall into class 1, the last two into class 2. Third, if we add more noise through \( A = A + \text{randn}(8,6) \) and \( B = B + \text{randn}(8,4) \), we obtain Eq. (11). Although the largest coefficients for the second and third new samples are not obvious, we are still able make the correct decision.

As the noise increases, the sparsity decreases in our examples. Therefore, incorrect decision may be made by the MAX rule. If we use the nearest-subspace rule instead, the residuals of \( B \) in Eq. (11) is shown in Table 1. According to the residuals, the first two samples are predicted in Class 1 and the last two samples are in Class 2.

(through \( B = B + 0.1 \times \text{randn}(8,4) \)), we would obtain Eq. (10). From the dominantly largest values in the columns of \( X \), we can predict that the first two new samples fall into class 1, the last two into class 2. Third, if we add more noise through \( A = A + \text{randn}(8,6) \) and \( B = B + \text{randn}(8,4) \), we obtain Eq. (11). Although the largest coefficients for the second and third new samples are not obvious, we are still able make the correct decision.

As the noise increases, the sparsity decreases in our examples. Therefore, incorrect decision may be made by the MAX rule. If we use the nearest-subspace rule instead, the residuals of \( B \) in Eq. (11) is shown in Table 1. According to the residuals, the first two samples are predicted in Class 1 and the last two samples are in Class 2.

(through \( B = B + 0.1 \times \text{randn}(8,4) \)), we would obtain Eq. (10). From the dominantly largest values in the columns of \( X \), we can predict that the first two new samples fall into class 1, the last two into class 2. Third, if we add more noise through \( A = A + \text{randn}(8,6) \) and \( B = B + \text{randn}(8,4) \), we obtain Eq. (11). Although the largest coefficients for the second and third new samples are not obvious, we are still able make the correct decision.

As the noise increases, the sparsity decreases in our examples. Therefore, incorrect decision may be made by the MAX rule. If we use the nearest-subspace rule instead, the residuals of \( B \) in Eq. (11) is shown in Table 1. According to the residuals, the first two samples are predicted in Class 1 and the last two samples are in Class 2.

(through \( B = B + 0.1 \times \text{randn}(8,4) \)), we would obtain Eq. (10). From the dominantly largest values in the columns of \( X \), we can predict that the first two new samples fall into class 1, the last two into class 2. Third, if we add more noise through \( A = A + \text{randn}(8,6) \) and \( B = B + \text{randn}(8,4) \), we obtain Eq. (11). Although the largest coefficients for the second and third new samples are not obvious, we are still able make the correct decision.

As the noise increases, the sparsity decreases in our examples. Therefore, incorrect decision may be made by the MAX rule. If we use the nearest-subspace rule instead, the residuals of \( B \) in Eq. (11) is shown in Table 1. According to the residuals, the first two samples are predicted in Class 1 and the last two samples are in Class 2.

<table>
<thead>
<tr>
<th>$b_1$</th>
<th>$b_2$</th>
<th>$b_3$</th>
<th>$b_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.5370</td>
<td>1.9087</td>
<td>4.4854</td>
<td>5.2291</td>
</tr>
<tr>
<td>8.3597</td>
<td>10.4016</td>
<td>2.2324</td>
<td>3.0576</td>
</tr>
</tbody>
</table>

Table 1

The residuals of \( B \) in Eq. (11).

Please cite this article as: Y. Li, A. Ngom, Classification approach based on non-negative least squares, Neurocomputing (2013), http://dx.doi.org/10.1016/j.neucom.2013.02.012
samples and four unknown samples is in Eq. (12), 

\[ D_{ij} = \begin{pmatrix} 2.7206 & 2.1430 & 0.1428 & 2.0088 & -0.0913 & 1.9284 \\ -1.9554 & 0.3454 & -1.9641 & -0.4381 & 1.2974 & 0.2921 \\ 5.5972 & 7.0132 & 6.7965 & -1.4534 & 0.1980 & 0.6267 \\ 1.1594 & 3.7783 & 2.7041 & -3.9547 & 2.1347 & -2.2222 \\ -0.9253 & -0.1314 & -0.7121 & -0.4530 & -0.0403 & -0.1376 \\ 3.8031 & 4.9755 & 4.3159 & 0.3239 & 0.8172 & 2.5853 \\ -1.7990 & -2.6933 & -3.7117 & 1.7895 & 2.8704 & 5.2992 \end{pmatrix} \]

First, the inputs are replaced with the precomputed inner products \( A^t A \) and \( A^t b \), and the computations \( A^t p \) and \( A^t b \) are equivalently replaced with \( (A^t A)p \) and \( (A^t b)p \), respectively. Second, predefined active set and passive set based on the previous iteration are given instead of starting from null passive set. Van Benthem and Keenan [37] proposed their FC-NNLS algorithm to further relieve the burden of FNLS. For the convenience of discussion, we also reformulated the pseudocode of FC-NNLS which is given in Algorithm 3.

**Algorithm 3. FCNNLS optimization algorithm.**

**Input:** matrices \( A_{m \times n} \), matrix \( B_{n \times q} \)

**Output:** matrix \( X_{n \times q} \) which is an optimal solution to 

\[ \min_{x \in \mathbb{R}^n} \frac{1}{2} \| B - AX \|_F^2 \text{ s.t. } A \geq 0 \]

\( Q = \{1 : q\}; \) % set of indices of columns of \( B \)

\( N = \{1 : n\}; \) % set of indices of rows of \( X \)

\( K = A^t A; \quad C = A^t B; \)

\( X = [K^{-1}C]_j; \) % use the unconstrained least squares solution to initialize \( X \), and \( X = [Y]_+ \) is defined as \( x_{ij} = y_{ij} \) if \( y_{ij} > 0 \), otherwise \( x_{ij} = 0 \)

\( R = \{c_j \mid c_j \neq 0, \max_{k \in P_i} h_{ik} > \epsilon, \mu_i = C^{-1}Kx_i, i \in Q\}; \) % set of indices of columns that to be optimized

**while** \( F \neq 0 \) **do**

\( J = \{c_j \mid c_j = \arg \max_{k \in P_i} h_{ik}, i \in F\}; \)

\( R_{F} = \{c_j \mid c_j = c_j - J, i \in F\}; \)

\( P_{F} = \{P_i \mid P_i = P_i + J, i \in F\}; \)

\( T = X_{F}; \)

\( T_{F} = \{T_i \mid T_i = \arg \min_{k} \frac{1}{2} \| B_i - A_{F}p \|_2^2\}; \)

**end while**

**while** \( r \neq 0 \) **do**

\( X_{F} = T; \)

\( J = \{c_j \mid c_j \neq 0, \max_{k \in P_i} h_{ik} > \epsilon, \mu_i = C^{-1}Kx_i, i \in Q\}; \)

**end while**

**Algorithm 4. CSSLS.**

**Input:** matrices \( K = A^t A \) and \( C = A^t B \), where \( A \) is of size \( m \times n \) and \( B \) is of size \( m \times l \), \( l \) passive sets \( \mathcal{P} \)

**Output:** set \( \mathcal{Y} = \{y_{i} \mid 1 \leq i \leq l, y_{i} = \arg \min_{x} \frac{1}{2} \| b_i - A_{F}x \|_2^2\} \) which can be denoted concisely by \( \mathcal{Y} = \arg \min_{x} \frac{1}{2} \| B - A_{F}z \|_2^2 \)

\( z = 0; \) % initialize

\( \mathcal{L} = \{1 : l\}; \) % set of indices of columns of \( X \)

\( \mathcal{U} = \text{unique} (\mathcal{P}); \)

\( \mathcal{E} = \{1 : \text{size}(\mathcal{U})\}; \) % set of indices of unique passive sets

\( \mathcal{S} = \{S_j \mid S_j = \{j \mid j \in \mathcal{L}, P_j = \mathcal{U}_i, i \in \mathcal{E}\}; \) % set of indices of columns of \( B \) sharing the identical passive sets

Please cite this article as: Y. Li, A. Ngom, Classification approach based on non-negative least squares, Neurocomputing (2013), http://dx.doi.org/10.1016/j.neucom.2013.02.012
for $\forall \ell \in \mathcal{U}$ do
\[ y_\ell = (K_{\ell, \cdot})^{-1}c_{\ell, :}; \]
end for

Algorithm 5. UR-NNLS optimization algorithm.

**Input:** matrices $A_{m \times n}$ and $B_{m \times p}$

**Output:** matrix $X$ which is a solution to
\[ \min_x \frac{1}{2} \| B - AX \|_F^2 \ 	ext{s.t.} \ \mathbf{x} \succeq 0 \]
\[ K = A^T A; \ C = A^T B; \ R = B^T B; \]
\[ X(X < 0) = 0; \]
\[ r_{\text{prev}} = \frac{1}{2} \| B - AX \|_F^2 = \frac{1}{2} tr(R - X^T C - C^T X + X^T KX); \]
\[ \text{for } i = 1 : \text{maxiter} \ 	ext{do} \]
\[ X_{ij} = X_{ij} + \frac{C_{ij}^2 + K_{ij} X_{ij}}{C_{ij} + K_{ij} X_{ij}}; \ % \text{define } M_{ji}^{(e)} = \frac{M_{ji}^{(e)} + M_{ij}^{(e)}}{2}; \]
\[ \text{if } i = \text{maxiter} \text{ or } i \mod m = 0 \ 	ext{then} \]
% check every l iterations if the termination conditions are met
\[ r_{\text{cur}} = \frac{1}{2} \| B - AX \|_F^2 = \frac{1}{2} tr(R - X^T C - C^T X + X^T KX); \]
\[ \text{if } r_{\text{prev}} - r_{\text{cur}} \leq \epsilon \text{ or } r_{\text{cur}} \leq \epsilon \ 	ext{then} \]
break;
\[ \text{end if} \]
\[ r_{\text{prev}} = r_{\text{cur}}; \]
end if
\[ \text{end for} \]

FC-NNLS has two improvements. First, the solution is initialized by the optimal-unconstrained-least-squares solution whose negative values are set to 0’s and positive variables are collected into passive sets. This initial solution approximates the optimal solution. Second, instead of solving the SLHS-NNLS problems sequentially, they are solved in a column-parallel strategy so as to share the common matrix inverses at each iteration. Eqs. (9) and (10), shown above, are two examples solved by FC-NNLS algorithm.

4.3.2. Iterative update rule algorithm

The second algorithm is based on multiplicative update rules. Multiple-update-rules algorithms have been popular for NMF since it was first developed by Lee and Seung [11,38]. The update rules are essentially gradient−descent rules. Two update rules are proposed by Ding et al. [19] to solve semi-NMF: $S = W H$. Semi-NMF is a relaxed NMF. Through semi-NMF, matrix $S$ can be decomposed into basis matrix $W$ and coefficient matrix $H$. Only $H$ is constrained by non-negativity. $W$ and $H$ are alternately updated by update rules. The update rule for $H$, fixing $W$, can be modified to solve our MLHS-NNLS problem. This is illustrated in Algorithm 5 where $W$ is fixed by $A$, and $H$ is redenoted by $X$.

Ding et al. [19] have proven that, when fixing $A$ and updating $X$ iteratively, the term $\| B - AX \|_F^2$ monotonically decreases. Based on this theory, we propose the update-rule-based NNLS (UR-NNLS) optimization algorithm which is shown in Algorithm 5. For indirect proof of convergence and time complexity of this algorithm, please refer to the work of Ding et al. [19]. The solution is also initialized by the unconstrained least squares solution as in Algorithm 3. Algorithm 5 does not always reach the optimal solutions, it provides a high-quality approximation to this problem and the classification task. Also, this algorithm is much easier to implement than active-set algorithms. Hereafter, we shall use the FC-NNLS algorithm as the default solver to our MLHS-NNLS problem.

We note that both FC-NNLS and UR-NNLS only use matrix inner products, $A^T A$, $A^T B$, and $B^T B$, in our formulation. This is because least squares is essentially a problem of quadratic programming. This property provides us two benefits. First, it helps us to handle missing values. Second, The NNLS based classification approaches can be extended to kernel version. We will introduce these two benefits in Sections 4.5 and 4.6.

4.4. Relation between sparsity and non-negativity

The optimal solution of the MLHS-NNLS problem is sparse, because in an optimal solution the components corresponding to the active set are 0, while the reminder corresponding to the passive set are positive. If the optimal solution to an unconstrained LS contains negative values, the optimal solution to the corresponding NNLS problem sits on the edge of the feasible regions. Hence large size of active set means the optimal solution is sparse. We can define a measure of sparsity as

\[ \text{sparsity}(X) = \frac{\delta(X < \epsilon)}{\text{row}(X) \times \text{column}(X)}, \]

where $\delta(M)$ is a function counting the number of non-zeros in $M$, $\epsilon$ is a small positive number (the default can be $10^{-4}$). For example, the sparsity of the coefficient matrices in Eqs. (10) and (11) are 0.5417 and 0.4167, respectively. It is well-known that the unconstrained least squares solution, which minimizes $l_2$-norm, is not sparse. Therefore, the high sparsity of our solution results from the non-negative constraint. Another example of this relation is SVM where the normal vector $w$ of separating hyperplane is a non-negative linear combination of all training samples. The coefficients are Lagrangian multipliers which are usually sparse. And the training samples corresponding to the non-zero coefficients are called support vectors.

4.5. Weighting strategy for missing values

We note that the two optimization algorithms described above only use the inner product of samples. Two inner product matrices are needed: $K = A^T A$ and $C = A^T B$ by both of Algorithms 3 and 5, where $A$ and $B$ represent the training data and unknown data with samples in columns, respectively. Additionally, Algorithm 5 also needs the inner product $R = B^T B$. If the samples are normalized to have unit $l_2$-norm, the inner product of two samples is actually the cosine similarity between them. Suppose we have two unnormalized samples $a$, $b \in \mathbb{R}^m$, their normalized inner product can be formulated as

\[ a^T b = \frac{a^T b}{\|a\|_2 \|b\|_2}. \]

This feature of these optimization algorithms is quite useful to handle missing values. If $a$ and $b$ have missing values, we can use the features without missing values in both of them to calculate the cosine similarity.

\[ a^T b = \frac{a^T b}{\|a\|_2 \|b\|_2}. \]

where $I$ is the set of indices of the features whose values are observed in both $a$ and $b$, that is $I = I_a \cap I_b$, where $I_a$ is the set of indices of features whose values are present in $a$. The difference between this weighting strategy with 0-imputation is that the inner product of a pair of 0-imputed samples is not the cosine similarity. If $a$ and $b$ are original samples with missing values imputed by 0, the normalized inner product of them is $a^T b = \frac{a^T b}{\|a\|_2 \|b\|_2}$. The performance of our weighting strategy and 0-imputation will be investigated in the experimental part.
Through this way, we can calculate the inner product matrices $K$, $C$, and $R$ in a pair-wise fashion, if $A$ and $B$ have missing values. The difference between this weighting strategy with the strategy of removing features with missing values as a preprocessing is that our strategy utilizes more information than the later. This is because our strategy computes the inner product of two samples based on their available features, while the feature pre-removal strategy does so based on the global available features. In some cases, the missing rate can be so large that all features are removed by the later strategy. But our pair-wise weighting strategy might still work normally. Therefore, the advantage of our strategy is that it can work in the cases of large missing rate.

If the NS rule is used after getting the sparse coefficient of each sample, the regression residual of each class can also be computed in this weighting strategy. From the following equation:
\[
\frac{1}{2} \| \mathbf{b} - \mathbf{A}_{\mathbf{x}i} \|_2^2 = \frac{1}{2} (\mathbf{b} - \mathbf{A}_{\mathbf{x}i})^T (\mathbf{b} - \mathbf{A}_{\mathbf{x}i}) = \frac{1}{2} \mathbf{b}^T \mathbf{b} - 2 \mathbf{b}^T \mathbf{A}_{\mathbf{x}i} + \mathbf{A}_{\mathbf{x}i}^T \mathbf{A}_{\mathbf{x}i},
\]
we can find that the residual of $\mathbf{b}$ regressed by the $i$-th class is a function of $\mathbf{b}^T \mathbf{b}$, $\mathbf{A}_{\mathbf{X}i}^T \mathbf{A}_{\mathbf{X}i}$, which can be computed by the weighting strategy in the case of missing values. $\mathbf{A}_{\mathbf{X}}$ is the training samples of the $i$-th class.

### 4.6. Kernel extension

Our NNLS approach described above may not work well on overcomplete data due to the following reasons. First, the systems of linear equations, $\mathbf{B} = \mathbf{A} \mathbf{X}$, may have one or multiple non-negative solutions. In this case, the solutions may not be necessarily sparse. Even if the solutions might be sparse, the unique solution returned by the optimization algorithms may not have the best semantic information. Second, the optimization algorithms need to compute the inverses of $\mathbf{K}$ and its submatrices, but they are singular matrices in this case. In the experimental section, we will show the performance of our NNLS approach on overcomplete data to verify the claim above. Because of the potential singularity problem when computing the matrix inverses, we can use two ways to approximate this. The first way is to add a very small identity matrices to the target matrix before computing the inverse.

\[
\mathbf{K}^{-1} \approx (\mathbf{K} + \gamma \mathbf{I})^{-1},
\]
where $\gamma$ is a very small positive number, for example, $2^{-32}$. The second way is to apply SVD.

\[
\mathbf{K}^{-1} \approx \mathbf{U} \mathbf{S}^{-1} \mathbf{V}^T,
\]
where $\mathbf{K} = \mathbf{U} \mathbf{S} \mathbf{V}^T$ and $f(s)$ is defined as $f(\lambda) = \lambda^{-1}$ if $\lambda \geq \gamma$; otherwise, $f(\lambda) = 0$. $\gamma$ is also a very small positive number. In fact, the solution using the first way is a solution to ridge regression. Ridge regression is a popular remedy to deal with the first reason above.

In order to improve the performance of our approach for overcomplete data, we have two options. One is to use $l_1$ - regularization on the coefficient vector in order to obtain more sparse result. Another is to kernelize them. Though the former is interesting, it has a problem of model selection. Therefore, we leave it for future investigation and focus on the later one. Let $\phi(\mathbf{b})$ be a mapping function which casts sample $\mathbf{b}$ to a feature space, we can define $\phi(\mathbf{B}) = [\phi(\mathbf{b}_1), \cdots, \phi(\mathbf{b}_p)]$. And the NNLS optimization in the feature space is

\[
\min_{\mathbf{X}} \frac{1}{2} || \phi(\mathbf{B}) - \phi(\mathbf{A}) \mathbf{X} ||^2_2 \quad \text{s.t.} \quad \mathbf{X} \geq 0.
\]

As has been mentioned in Section 4.5, the learning and prediction of our NNLS approach only involves the inner products $\mathbf{K} = \langle \phi(\mathbf{A})^T \phi(\mathbf{A}) \rangle$, $\mathbf{C} = \langle \phi(\mathbf{A})^T \phi(\mathbf{B}) \rangle$, and $\mathbf{R} = \langle \phi(\mathbf{B})^T \phi(\mathbf{B}) \rangle$. Therefore, we can use kernel trick to compute these three matrices: $\mathbf{K} = k(\mathbf{A}, \mathbf{A})$, $\mathbf{C} = k(\mathbf{A}, \mathbf{B})$, and $\mathbf{R} = k(\mathbf{B}, \mathbf{B})$, where $k(\mathbf{A}, \mathbf{B})_{ij} = k(\mathbf{a}_i, \mathbf{b}_j) = (\phi(\mathbf{a}_i))^T \phi(\mathbf{b}_j)$.

Through this way, FC-NNLS and UR-NNLS are extended to kernel FC-NNLS (FC-KNNLS) and kernel UR-NNLS (UR-KNNLS). The corresponding extension of classification algorithms are KNNLS-MAX and KNNLS-NS. The performance of them are explored in the experimental part. In order to build robust optimization algorithms for kernel NNLS methods, Eqs. (16) and (17) are also applied to compute matrix inverses.

Model selection for kernel approaches based on $k$-fold CV accuracy can be expensive in practice, because for each point of parameter, a classifier has to run for $k$ times. Therefore, we propose a randomized KNNLS method to avoid model selection. The idea is in the following. Bootstrapping is rerun for $b$ times to generate $b$ training sets. At each execution, about 66.7% training samples are selected as input of our kernel NNLS classifier, and the kernel parameter is randomly selected from a specific range. Finally, the class labels are voted over the $b$ results by the majority rule. This method is coined bootstrapping NNLS (BNNLS) method. If the MAX (or nearest-subspace) rule is used by each kernel NNLS classifier, it is denoted as BNNLS-MAX (or BNNLS-NS). The algorithm is shown in Algorithm 6.

**Algorithm 6.** Bootstrapping kernel non-negative least squares classifier.

Input: $A_{m,n}$: training set including $m$ features and $n$ samples, $c$: class labels of the training samples, $B_{m,p}$: $p$ new samples, $b$: the number of classifiers

Output: $p$: predicted class labels of the $p$ new samples

for $t = 1$ to $b$

- $T = \text{bootstrap}(A)$;
- take kernel parameter randomly: $\sigma = \text{rand}(r)$; $\% r$ is a specified set of real numbers or an interval
- $p = \text{KNNLS}(T, c, B, \sigma)$;
- $P(:, t) = p$;
end
for $p = \text{vote}(P)$;

The function $\text{bootstrap}(A)$ randomly selects 66.7% samples from $A$. $\text{rand}(r)$ choose a value as kernel parameter from the specific range $r$. $\text{vote}(P)$ is a function that makes the final decision based on the decisions of the classifier committee by majority rule.

### 4.7. Kernel NNLS for tensor data

Our NNLS classifier can be extended to classify tensor data. Interested readers are referred to the review of Kolda and Bader [4] for tensor algebra. Without the loss of generality, we suppose there are $l_3$ training samples represented by an order-3 tensor $\mathbf{A}_{l_1 \times l_2 \times l_3}$. The third mode is the class axis. Each sample is a matrix of size $l_1 \times l_2$. Therefore, $\mathbf{A}(; :, i)$ is the $i$-th training sample. Suppose we have $p$ new samples in $\mathbf{B}_{l_1 \times l_2 \times p}$. Assume that each of such new samples can be regressed by a non-negative linear combination of the training samples. We need to solve the following NNLS problem:

\[
\min_{\mathbf{X}} \frac{1}{2} \| \mathbf{B} - \mathbf{A} \times_3 \mathbf{X} \|_F^2 \quad \text{s.t.} \quad \mathbf{X} \geq 0,
\]

where $\times_3$ is the mode-3 product [4] as defined at the end of Section 1. Through matricizing tensors to matrices, we can convert the above optimization task into the equivalent formula:

\[
\min_{\mathbf{X}} \frac{1}{2} \| \mathbf{B}_{(3)} - \mathbf{A}_{(3)} \mathbf{X} \|_F^2 \quad \text{s.t.} \quad \mathbf{Y} \geq 0,
\]

where $\mathbf{A}_{(3)}$ is a matrix of $l_3 \times (l_1 \times l_2)$, unfolded from tensor $\mathbf{A}$ in mode 3 which is also defined at the end of Section 1.
Using transposition, we have
\[
\min_{\mathbf{X}} \frac{1}{2} \| \mathbf{B}^\top \mathbf{A} \mathbf{X} - \mathbf{A}^\top \mathbf{Y} \|^2 \quad \text{s.t. } \mathbf{X} \succeq 0.
\] (21)

Now Eq. (21) can be solved by FC-NNLS and Algorithm 5. Our NNLS classifier, described in Algorithm 2, can now be generalized for tensor data.

The drawback of this generalization is that the structural information within a sample is not considered. The objective (Eq. (19)) uses the Euclidean distance, hence the samples are actually vectorized in Eqs. (20) and (21). If we use other dissimilarity or similarity metric which takes the temporal information into account when classifying gene-sample-time data, the performance is expected to be increased. We propose to apply dynamical-systems kernel [39] in our kernel NNLS method for GST data. Dynamical systems kernel accepts matrix inputs and takes into account the temporal information.

### 5. Experiments

In this section, we explore the performance of NNLS methods in various aspects including the prediction capability, the robustness to missing values and noise, time-complexity, and sparsity of coefficient generated by our method. We conducted experiments over 28 datasets which are briefly described in Table 2. We compared our methods with diverse benchmark methods including sparse representation, instance-based, and kernel methods. They are summarized in Table 3.

#### 5.1. Estimation of data size requirement

The statistical methodology using learning curve [57] is to use available classification results to estimate sample-size requirement for future experiments and evaluate the benefit in performance with extra samples. It involves two tasks. First of all, the minimum sample size, where significant accuracy is obtained, is estimated. Second, larger sample sizes and the corresponding classification error rates are used to fit the learning curve. The learning curve is an inverse power-law model expressed as \( e(n) = an^{-x} + b \), where \( a \) is the learning rate, \( x \) is the decay rate, and \( b \) is an asymptotical Bayes error. In order to know how the performance of NNLS changes with the size of training samples, we estimated (1) the minimum data sizes required for significant accuracy and (2) the learning curves of our NNLS over eight microarray gene expression datasets.

We implemented the method proposed in [57]. For each sample size, we repeated the partition of training and test set 50 times. For each split, the number of permutation is 50. We set the significant level to 0.05. The comparison with SVM is summarized in Table 4. As an example, the learning curves of NNLS and SVM over ALL are plotted in Fig. 1.

First of all, from Table 4, it can be clearly observed that, the minimum sample size (required to obtain significant accuracy) of NNLS is consistently smaller than SVM. It may be because NNLS belongs to instance-based learning while SVM needs sufficient training samples to learn the model parameter (that is the normal vector of hyperplane) for all future prediction. Second, from Table 4 and Fig. 1, we can find that the learning rate of SVM is consistently faster than NNLS. Third, the average error rate of NNLS is smaller than SVM on five datasets. Considering both error rate and \( p \) value, it is hard to find the difference of their

---

**Table 3**

The summary of our NNLS methods and the benchmark methods.

<table>
<thead>
<tr>
<th>Category</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our</td>
<td>NNLS-MAX</td>
<td>NNLS classifier with MAX rule as sparse interpreter. Linear kernel is used for overcomplete data, and radial basis function (RBF) kernel is used for undercomplete data</td>
</tr>
<tr>
<td></td>
<td>NNLS-NS</td>
<td>NNLS classifier with nearest-subspace rule as sparse interpreter. Linear kernel is used for overcomplete data, and RBF kernel is used for undercomplete data</td>
</tr>
<tr>
<td></td>
<td>BNNSLS-NS</td>
<td>Bootstrapping NNLS classifier with nearest-subspace rule as sparse interpreter. RBF kernel is used</td>
</tr>
<tr>
<td>SR</td>
<td>SRC</td>
<td>SRC (Eq. (3)) is used for undercomplete data, SRC1 (Eq. (2)) is used for overcomplete data</td>
</tr>
<tr>
<td></td>
<td>MSRC</td>
<td>The metasample-based sparse representation classification approach proposed in [30]</td>
</tr>
<tr>
<td>Instance</td>
<td>LRC</td>
<td>Linear regression classification approach, an independent modelling method proposed in [22]</td>
</tr>
<tr>
<td></td>
<td>1-NN</td>
<td>1-nearest neighbor approach</td>
</tr>
<tr>
<td>Kernel</td>
<td>SVM</td>
<td>Linear kernel is used for high-dimensional data and RBF kernel is used for low-dimensional data</td>
</tr>
<tr>
<td></td>
<td>ELM</td>
<td>Extreme learning machine proposed in [56]</td>
</tr>
</tbody>
</table>

---

Please cite this article as: Y. Li, A. Ngom, Classification approach based on non-negative least squares, Neurocomputing (2013), [http://dx.doi.org/10.1016/j.neucom.2013.02.012](http://dx.doi.org/10.1016/j.neucom.2013.02.012)
Table 4

<table>
<thead>
<tr>
<th>Data</th>
<th>Method</th>
<th>Min. size</th>
<th>Max. size</th>
<th>Error rate (p-value)</th>
<th>25th quantile</th>
<th>Mean</th>
<th>75th quantile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adenoma</td>
<td>NNLS-NS</td>
<td>6</td>
<td>25</td>
<td>0.02(1.12e–2)</td>
<td>(0.15,0.73,0.00)</td>
<td>(0.60,1.31,0.02)</td>
<td>(1.85,1.80,0.02)</td>
</tr>
<tr>
<td></td>
<td>SVM</td>
<td>13</td>
<td>25</td>
<td>0.03(2.12e–2)</td>
<td>(99.80,2.99,0.00)</td>
<td>(78.41,2.61,0.00)</td>
<td>(5.79,2.28,0.07)</td>
</tr>
<tr>
<td>Breast</td>
<td>NNLS-NS</td>
<td>10</td>
<td>60</td>
<td>0.14(1.20e–3)</td>
<td>(1.24,0.59,0.00)</td>
<td>(1.15,0.50,0.00)</td>
<td>(1.16,0.45,0.00)</td>
</tr>
<tr>
<td></td>
<td>SVM</td>
<td>11</td>
<td>60</td>
<td>0.25(1.16e–2)</td>
<td>(2.37,0.53,0.01)</td>
<td>(1.87,0.44,0.00)</td>
<td>(1.41,0.34,0.00)</td>
</tr>
<tr>
<td>Colon</td>
<td>NNLS-NS</td>
<td>8</td>
<td>55</td>
<td>0.11(2.00e–2)</td>
<td>(0.45,0.40,0.00)</td>
<td>(0.39,0.25,0.00)</td>
<td>(0.45,0.26,0.00)</td>
</tr>
<tr>
<td></td>
<td>SVM</td>
<td>14</td>
<td>55</td>
<td>0.08(9.60e–3)</td>
<td>(2.30,0.97,0.00)</td>
<td>(2.15,0.77,0.00)</td>
<td>(9.26,1.30,0.08)</td>
</tr>
<tr>
<td>Leukemia</td>
<td>NNLS-NS</td>
<td>6</td>
<td>60</td>
<td>0.05(8.00e–4)</td>
<td>(1.38,1.13,0.00)</td>
<td>(2.33,1.25,0.03)</td>
<td>(4.01,1.54,0.07)</td>
</tr>
<tr>
<td></td>
<td>SVM</td>
<td>8</td>
<td>60</td>
<td>0.02(0)</td>
<td>(16.36,2.07,0.00)</td>
<td>(42.54,2.28,0.03)</td>
<td>(100.00,2.45,0.05)</td>
</tr>
<tr>
<td>ALL</td>
<td>NNLS-NS</td>
<td>6</td>
<td>100</td>
<td>0.10(0)</td>
<td>(2.49,0.85,0.04)</td>
<td>(2.31,0.77,0.03)</td>
<td>(2.19,0.69,0.03)</td>
</tr>
<tr>
<td></td>
<td>SVM</td>
<td>13</td>
<td>100</td>
<td>0.15(0)</td>
<td>(5.30,0.77,0.00)</td>
<td>(4.46,0.69,0.00)</td>
<td>(4.09,0.65,0.00)</td>
</tr>
<tr>
<td>Breast5</td>
<td>NNLS-NS</td>
<td>5</td>
<td>100</td>
<td>0.22(0)</td>
<td>(0.51,0.39,0.10)</td>
<td>(0.77,0.69,0.18)</td>
<td>(1.48,1.07,0.23)</td>
</tr>
<tr>
<td></td>
<td>SVM</td>
<td>16</td>
<td>100</td>
<td>0.18(0)</td>
<td>(2.50,0.63,0.01)</td>
<td>(2.51,0.61,0.01)</td>
<td>(2.51,0.63,0.05)</td>
</tr>
<tr>
<td>MLL</td>
<td>NNLS-NS</td>
<td>4</td>
<td>60</td>
<td>0.07(0)</td>
<td>(0.50,0.62,0.00)</td>
<td>(0.87,1.07,0.07)</td>
<td>(1.69,1.52,0.10)</td>
</tr>
<tr>
<td></td>
<td>SVM</td>
<td>16</td>
<td>60</td>
<td>0.09(1.60e–3)</td>
<td>(1.78,0.56,0.00)</td>
<td>(3.16,0.66,0.00)</td>
<td>(4.88,0.74,0.00)</td>
</tr>
<tr>
<td>SRBCT</td>
<td>NNLS-NS</td>
<td>4</td>
<td>60</td>
<td>0.01(2.40e–2)</td>
<td>(2.10,1.39,0.00)</td>
<td>(1.63,1.06,0.00)</td>
<td>(1.66,0.96,0.00)</td>
</tr>
<tr>
<td></td>
<td>SVM</td>
<td>19</td>
<td>60</td>
<td>0.03(8.00e–4)</td>
<td>(100.00,1.87,0.00)</td>
<td>(100.00,1.82,0.00)</td>
<td>(100.00,1.79,0.00)</td>
</tr>
</tbody>
</table>

Fig. 1. The learning curve of NNLS and SVM on ALL. The sample sizes to fit the learning curves are [6, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100] and [13, 20, 30, 40, 50, 60, 70, 80, 90, 100], respectively for NNLS and SVM.

5.2. On undercomplete data

In order to know the performance of our NNLS approaches on undercomplete data, we compared our NNLS approaches with SRC, MSRC, LRC, 1-NN, SVM, and ELM on eight microarray data and five mass spectrometry data. We used the Friedman test with Nemenyi test as post hoc test [58]. This non-parametric statistical methodology is recommended in Demšar [58], for comparing multiple classifiers (> 5) over multiple datasets (> 10), because it is simple, safe, and robust. It involves two steps. First, the average ranks of classifiers over all datasets are computed, and Friedman test is conducted to verify the null-hypothesis that all classifiers are equivalent in the respect of classification performance. If the null hypothesis is rejected, a post hoc test (for example, Nemenyi test) is continued. If the average ranks of two classifiers differ by at least the critical difference (CD), then it can be concluded that their performance are significantly different. In our experiment, the average accuracies (of 4-fold cross-validation) of all classifiers over each dataset is obtained. All classifiers ran on the same partition of training and test sets. 4-fold cross-validation was repeated for 50 and 20 times on microarray and mass spectrometry data, respectively. We set the significant level to \( \alpha = 0.05 \). The CD diagram is illustrated in Fig. 2.

The bar plot of the average accuracy over five datasets is shown in Fig. 3, from which we can see that good result can be obtained by our NNLS approaches. In Fig. 2, we can identify two groups. The first group includes NNLS-NS, SVM, and SRC. The secondary group is composed of MSRC, NNLS-MAX, 1-NN, and ELM. Through statistical comparison, we can find that: (i) NNLS-NS is significantly better than NNLS-MAX on undercomplete data; (ii) NNLS-NS can obtain state-of-the-art result, and there is no significant difference among NNLS-NS, SVM, and SRC in such case.

5.3. Handle missing values

For the purpose of exploring the capability of our weighting strategy of handling missing values, we did experiment on all microarray data with randomly and artificially generated missing values. The missing rate ranges from 0.1 to 0.7. We used 4-fold cross-validation for 50 times at each missing rate for each dataset. We compared our strategy with 0-imputation. 0-imputation is the strategy to fill the missing values by constant 0’s. We did not conduct k-NN imputation [32], because for large missing rate, there is no complete feature and sample, and k-NN imputation can thus fail for this case. As an example, the average accuracies over SRBCT are plotted in Fig. 4.
employed Friedman test on all microarray data with missing rates 60% and 70% (therefore 16 datasets). We set the significant level to \( \alpha = 0.05 \). In our experiment, the null hypothesis was rejected, and the result of Nemenyi test is shown in Fig. 5.

First of all, we can see that NNLS-MAX using weighting strategy is significantly better than its counterpart using 0-imputation. If we increase the threshold of Type-I error slightly, then NNLS-NS using weighting strategy also performs significantly better than its counterpart using 0-imputation. Therefore it can be concluded that the weighting strategy is significantly better than 0-imputation when using the same classification technique. Second, from Fig. 4 we can see that NNLS-NS using weighting strategy and SVM using 0-imputation still maintain high accuracy at severe missing rate. And from Figure 5, we can find both of them obtained top ranks. It hence can prove the robustness of our weighting strategy.

5.4. On facial data

Now, we investigate the robustness of NNLS to corruption on facial data. SRC1 and LRC were proposed specifically for face recognition. Our proposed approach competed with them and others over three facial datasets (face95, face96, and grimace) collected by Spacik [52]. It was mentioned by Spacik [52] that it is difficult to classify face96 and grimace due to variation of background and scale, and extreme variation of expression. We randomly picked up 20 individuals for face95 and face96, respectively, and used the whole grimace data in our experiment. As was done by Wright et al. [15] and Naseem et al. [22], downsampling was employed for SRC1 and LRC with downsampling rate 1/16. We artificially replaced 10%–70% of the pixel values with random values from the range of [0,255]. The comparison with respect to the mean accuracy over 20 runs of 4-fold cross-validation on face95 is shown in Fig. 6.
It can be seen that NNLS, SRC, and MSRC look very robust to random corruption. In order to verify this intuition, we applied Friedman test on these datasets with corruption rates from 10% to 70% (thus 21 datasets). The significant level is again 0.05. The null hypothesis is rejected. The result of Nemenyi test is given in Fig. 7.

We can identify two groups of classifiers: the robust one including SRC, MSRC, and NNLS; and the sensitive one including LRC, SVM, and ELM.

5.5. On overcomplete data

We did two parts of experiments on overcomplete data. First, in order to see the performance of our NNLS method (with linear kernel) on overcomplete data, we compared it with SRC on LM dataset. The result is in Table 5.

We can see that the performance of our NNLS method is worse than SRC. The accuracies of NNLS-NS are 0.7760 and 0.7969 using FC-NNLS and UR-NNLS algorithms, respectively, while SRC obtained 0.8665. This result corroborates our claim that the NNLS methods with linear kernel may not work well on overcomplete data.

Next, for the purpose of exploring the performance of kernel NNLS methods (NNLS-MAX, NNLS-NS, and BNNLS-NS), we compared them with benchmark methods on 6 binary-class and 5 multi-class low dimensional datasets. These data are described in the second last block of Table 2, where BCWD stands for Breast Cancer Wisconsin Diagnostic, BSTT for Breast Tissue, and LM for Libras Movement. Our NNLS methods used the well-known RBF kernel. The benchmark methods includes SRC, LRC, 1-NN, SVM, and ELM using RBF as activation function. In order to maximize the capability of the methods, model selection was employed. The optimal kernel parameters of NNLS-MAX, NNLS-NS were obtained by line search with the average accuracy of inner 3-fold cross-validation on training data (a higher folds of cross-validation is not computationally affordable). The superparameters and kernel parameters of SVM were attained through grid search with inner 3-fold cross-validation on training set. It was claimed by Wang and Huang [56] that the performance of ELM is not sensitive to the number of hidden nodes. Thus it was set to 3 times as the number of training samples. Using outer 4-fold cross-validation, all the algorithms were executed on the same partition of training and test sets, and it is repeated for 20 times. The average accuracies of five out of eleven datasets are illustrated in Fig. 8.

Comparing the results on LM in Table 5 and Fig. 8, we can see that the accuracy of NNLS is increased by around 10% using RBF kernel.
We resorted to Friedman test ($\alpha = 0.05$) again to compare the classifiers over these eleven datasets. The null hypothesis is rejected, and the post hoc Nemenyi test is presented graphically in Fig. 9.

From this diagram, we can see that NNLS-NS obtained the best average rank and BNNLS-NS has very close average rank as SVM. Our statistical test implies that there is no significant difference between NNLS-NS and BNNLS-NS. This convinces us that BNNLS-NS is an effective method to avoid model selection. The reason for this may be because the minimum sample size required by NNLS-NS for significant accuracy is usually very small. The Nemenyi test also suggests no critical difference between NNLSs and SVM.

5.6. On all datasets

We have investigated the performance of NNLS methods on several situations separately so far. Now we compared them with benchmark approaches using the results of all 27 datasets including microarray data, mass spectrometry data, facial data, and low dimensional data. We conducted Friedman test with significant level $\alpha = 0.05$. The null hypothesis is rejected and the Nemenyi test is presented in Fig. 10.

Two groups of classifiers can be identified. NNLS-NS, SVM, SRC form the first group which obtain top ranks. And they do not differ significantly. The second group is composed of NNLS-MAX, 1-NN, LRC, ELM, and MSRC.

5.7. Compare computing time

In order to investigate the time-complexity of our method on undercomplete data, we record the averaged computing time

<table>
<thead>
<tr>
<th>Table 5</th>
<th>Mean accuracy on LM data of 20 runs of 4-fold cross-validation.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimization</td>
<td>Method</td>
</tr>
<tr>
<td>FC-NNLS</td>
<td>NNLS-MAX</td>
</tr>
<tr>
<td></td>
<td>NNLS-NS</td>
</tr>
<tr>
<td>UR-NNLS</td>
<td>NNLS-MAX</td>
</tr>
<tr>
<td></td>
<td>NNLS-NS</td>
</tr>
<tr>
<td>$l_1$-magic</td>
<td>SRC</td>
</tr>
</tbody>
</table>
including training time on 3/4 of data and test time on the rest. We compared our method with others on one binary-class and one multi-class microarray datasets (Colon and Breast5), one mass spectrometry dataset (PC-IMAC-Cu), and one facial dataset (face95). The result is shown in Fig. 11.

First, we can see that the NNLS approaches run faster than SRC. On face5, SRC has similar running time because features are downsampled when using SRC. Second, SVM is slower than NNLS methods over face95. This implies that SVM is slowed down, while the NNLS approaches are more efficient, in the case of large number of classes.

We also compared the computing time of BNNLS-NS with NNLS with model selection over Parkinsons and BSTT which are overcomplete data. The average of computing time, including model selection, training and test, is illustrated in Table 6.

From Table 6 we can observe that our BNNLS approach is slightly efficient than NNLS with model selection. In fact, the efficiency of BNNLS is determined by the number of repetition of bootstrapping. In current study, we fix it to 100. In the future, we will investigate the smallest number of repetition in BNNLS where equivalent accuracy can be obtained compared with NNLS using model selection.

5.8. On tensor data
We tested our approach on a tensor dataset of size 53 (genes) x 7 (time-points) x 27 (samples), which is denoted as INFb (Table 2). We compared our NNLS methods with hidden Markov models (HMMs) [59], SVM with dynamical-systems kernel [39], and high-order NMF [3]. We applied 9-fold cross-validation to split the dataset into training set and test set (because there are only 27 samples and in each fold there are right 3 samples in the test set). It was repeated 50 times for NNLS methods. And the results of HMMs, SVM, and HONMF are obtained from Li and Ngom [3]. 9-fold cross-validation was repeated 20 times for these benchmark methods, as a large number of repetition is computationally unaffordable. The preliminary result are shown in Table 7.

Instead of conducting strict statistical test, we would rather treat these results just as example of applying NNLS on tensor data, because we do not have enough tensor data. First, from the comparison between NNLS using linear kernel and its counterpart using dynamical-systems kernel, we can see that the later obtained better result, because it considers the temporal
information within the samples. Hence, the structural information within samples does contribute the discrimination. Second, using dynamical-systems kernel, BNNLS-NS obtained comparable accuracy with NNLS-NS. It corroborates that BNNLS-NS is an effective method avoiding model selection. Third, kernel NNLS-NS has comparable accuracy with HMMs and SVM, however it runs much faster than these benchmark methods. These preliminary results convince us that NNLS methods with appropriative kernels, which take the specific structural information into account, perform well in the aspect of prediction capability and time-complexity. Thus they are worthy of further investigation for tensor data.

Table 6
Computing Time (in Seconds) With Model Selection versus Without Model Selection. These are mean results of 20 runs of 4-fold cross-validation.

<table>
<thead>
<tr>
<th>Method</th>
<th>Parkinsons BSTT</th>
<th>Parkinsons BSTT</th>
</tr>
</thead>
<tbody>
<tr>
<td>NNLS-MAX</td>
<td>4.319</td>
<td>1.498</td>
</tr>
<tr>
<td>NNLS-NS</td>
<td>4.431</td>
<td>1.546</td>
</tr>
<tr>
<td>BNNLS-NS</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

Table 7
Classification Performance on INFb Data. BACC stands for balanced accuracy, that is 0.5(sensitivity + specificity). STD stands for standard deviation. The unit of time is second. The values in this table are the average results over 20 runs of 9-fold cross-validation.

<table>
<thead>
<tr>
<th>Data</th>
<th>Methods</th>
<th>Accuracy (STD)</th>
<th>BACC (STD)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>INFb</td>
<td>NNLS-MAX</td>
<td>0.588 (0.039)</td>
<td>0.554 (0.043)</td>
<td>0.005</td>
</tr>
<tr>
<td></td>
<td>NNLS-NS</td>
<td>0.744 (0.024)</td>
<td>0.663 (0.022)</td>
<td>0.005</td>
</tr>
<tr>
<td></td>
<td>kernel NNLS-NS</td>
<td>0.761 (0.043)</td>
<td>0.754 (0.043)</td>
<td>0.470</td>
</tr>
<tr>
<td></td>
<td>kernel BNNLS-NS</td>
<td>0.788 (0.029)</td>
<td>0.777 (0.030)</td>
<td>10.571</td>
</tr>
<tr>
<td></td>
<td>HMMs</td>
<td>0.761 (0.047)</td>
<td>0.711 (0.038)</td>
<td>2117</td>
</tr>
<tr>
<td></td>
<td>SVM</td>
<td>0.789 (0.023)</td>
<td>0.697 (0.019)</td>
<td>93</td>
</tr>
<tr>
<td></td>
<td>HONMF</td>
<td>0.815 (0.040)</td>
<td>0.828 (0.065)</td>
<td>1662</td>
</tr>
</tbody>
</table>

Fig. 10. Graphical representation of Nemenyi test on all datasets.

Fig. 11. Computing time on diverse datasets. The unit is second. These are mean results of 50 and 20 runs of 4-fold cross-validation for microarray and other data, respectively. Logarithm of base 2 was taken for better comparison.
5.9. Sparsity

We recorded the sparsity of NNLS and SRC on each dataset. And the sparsity of some datasets are given in Fig. 12. It can be seen that NNLS can obtain very sparse coefficients. So far, we have not yet find any case where NNLS fails to obtain sparse result.

6. Conclusion and future work

In this study, we propose a novel classifier based on non-negative least squares to classify undercomplete data. For a new sample, our approach pursues a non-negative and sparse linear combination of training samples. Two rules are applied as sparse interpreters in our approach to predict new samples according to the sparse coding. A weighting strategy is proposed to handle missing values. And our approach is kernelized to classify overcomplete data. An ensemble approach based on bootstrapping is proposed to avoid model selection. And our approach is also applied to classify tensor data by an appropriate kernel. Intensive experiments using statistical test have shown that our approach and its extensions perform very well in various cases. Using learning curves, it revealed that NNLS has a smaller minimum sample size than SVM where significant accuracy is obtained. Friedman test with post hoc Nemenyi test indicates that there is no significant difference between NNLS and SVM. We also show that our NNLS using weighting strategy is very robust to missing values. Our NNLS approach is also as robust as SRC to random corruption over facial data. Comparison result over overcomplete data proves that the bootstrapping based NNLS is an effective methods to avoid model selection. Comparison over various data shows that our NNLS approaches are very efficient. Furthermore, preliminary result shows that our approach works very well and efficiently on tensor data. Experimental result shows that very sparse coefficient coding can be obtained by the NNLS optimization.

It could be beneficial to apply our NNLS approach to both large-scale undercomplete and overcomplete data which may be subject to serious missing values or noise. It is also promising to classify tensor data using our approach with appropriate kernels. The high sparsity can help users to interpret the result well. We have developed the Sparse Representation Toolbox in MATLAB. This toolbox includes our NNLS approach as well as SRC1, SRC2, MSRC, and LRC. The implementation of statistical tests used in this study are also included. It is available at http://cs.uwindsor.ca/~li11112c/sr. The necessity of combining non-negative constraint with $l_1$-norm minimization will be explored. The NNLS optimization algorithms will also be improved to classify data with large number of samples. Dictionary learning methods might be considered for this case. Since NNLS has a very small minimum data size requirement for significant accuracy, we will explore its performance for imbalanced data. The effect of normalization will be studied for NNLS. New kernels will be investigated to measure the similarity of two samples for tensor data.

Acknowledgments

We would like to thank the reviewers who gave us valuable suggestions on the presentation and statistical comparison in this article. This study has been supported by IEEE CIS Summer Research Grant 2010, Ontario Graduate Scholarship 2011–2013, and Canadian NSERC Grants #RGPIN228117-2011.

References


C.L. Lawson, R.J. Hanson, Solving Least Squares Problems, SIAM, Philadelphia.


