Chapter 2

Sparse Representation for High-Dimensional Data Analysis

1 Introduction

The studies in biology and medicine have been revolutionarily changed since the inventions of many high-throughput sensory techniques. Using these techniques, the molecular phenomena can be probed with a high resolution. In the virtue of such techniques, we are able to conduct systematic genome-wide analysis. In the last decade, many important results have been achieved by analyzing the high-throughput data, such as microarray gene expression profiles, gene copy numbers profiles, proteomic mass spectrometry data, next-generation sequences, and so on.

On one hand, biologists are enjoying the richness of their data; one another hand, bioinformaticians are being challenged by the issues of high-dimensional data as well as by the complexity of bio-molecular data. Many of the analysis can be formulated as machine learning tasks. First of all, we have to face the cures of high dimensionality, which means that many machine learning models are unable to correctly predict the classes of unknown samples due to the large number of features and the small number of samples in such data. In other words, the machine learning models can be overfitted and therefore have poor capability of generalization. Second, if the learning of a model is sensitive to the dimensionality, the learning procedure could be extremely slow. Third, many of the data are very noisy, therefore the robustness of a model is necessary. Fourth, the high-throughput data exhibit a large variability and redundancy, which make the mining of useful knowledge difficult. Moreover, the observed data usually do not tell us the key points of the story. We need to discover and interpret the latent factors which drive the observed data.

Many of such analyses are classification problems from the machine learning viewpoint. Therefore in this study, we focus our study on the classification techniques for high-dimensional biological data. The machine learning techniques addressing the challenges above can be categorized into two classes. The first one aims to directly classify the high-dimensional data while keeping a good generalization ability and efficiency in

optimization. The most popular method in this class is the *regularized basis-expanded linear model*. The regularization aims to avoid overfitting a model by reducing the model complexity while fitting it. One example is the state-of-the-art *support vector machines* (SVM) [5]. SVM is sparse linear model that uses hinge loss and l_2-norm regularization. It can be kernelized and its result is theoretically sound. Combining different regularization terms and various loss functions, we can have many variants of such linear models [6]. In addition to classification, some of the models can be applied to regression and feature (bio-marker) identification. However, most of the learned linear models are not interpretable, while interpretability is usually the requirement of biological data analysis. Moreover, linear models can not be extended *naturally* to multi-class data, while in bioinformatics a class may be composed of many subtypes. See Chapter 5 for a through discussion on linear models.

Another technique of tackling with the challenges above is dimensionality reduction which includes feature extraction and feature selection. *Principal component analysis* (PCA) [7] is one of the traditional feature extraction methods and is widely used in processing high-dimensional biological data. The basis vectors produced by PCA are orthogonal, however many patterns in bioinformatics are not orthogonal at all. The classic *factor analysis* (FA) [8] also has orthogonal constraints on the basis vectors, however its Bayesian treatment does not necessarily produce orthogonal basis vectors. Bayesian factor analysis will be introduced in the next section.

Sparse representation (SR) [9] is a parsimonious principle that a sample can be approximated by a sparse linear combination of basis vectors. Non-orthogonal basis vectors can be learned by SR, and the basis vectors may be allowed to be redundant. SR highlights the parsimony in representation learning [10]. This simple principle has many strengthes that encourage us to explore its usefulness in bioinformatics. First, it is very robust to redundancy, because it only select few among all of the basis vectors. Second, it is very robust to noise [11]. Furthermore, its basis vectors are non-orthogonal, and sometimes are interpretable due to its sparseness [12]. There are two techniques in SR. First, given a basis matrix, learning the sparse coefficients of a new sample is called *sparse coding*. Second, given training data, learning the basis vector is called *dictionary learning*. As dictionary learning is, in essence, a sparse matrix factorization technique. *Non-negative matrix factorization* (NMF) [13] can be viewed a specific case of SR. For understanding sparse representation better, we will give the formal mathematical formulation from a Bayesian perspective in the next section.

In this chapter, we investigate sparse representation for high-dimensional data analysis systematically. First, we formulate sparse representation from a Bayesian viewpoint. Then, we discuss sparse coding based classification and the corresponding optimization algorithms. After that, we investigate dictionary learning based dimension reduction techniques. We shall present a generic dictionary learning framework and a more general model called versatile sparse matrix factorization. We then propose a supervised dictionary learning method for multi-class data. We also propose a novel classification approach, called clustering-and-classifying, that combines both dictionary learning and sparse coding.

2 Bayesian Sparse Representation

Both (sparse) factor analysis and sparse representation models can be used as dimension reduction techniques. Due to their intuitive similarity, it is necessary to give their definitions for comparison. In this section, we briefly survey the sparse factor analysis and sparse representation in a Bayesian viewpoint. The introduction of sparse representation is helpful to understand the content of the subsequent sections. Hereafter, we use the following notations unless otherwise stated. Suppose the training data is $\boldsymbol{D} \in \mathbb{R}^{m \times n}$ (m is the number of features and n is the number of training instances (samples or observations)), the class information is in the n-dimensional vector $\boldsymbol{c}$. Suppose p new instances are in $\boldsymbol{B} \in \mathbb{R}^{m \times p}$.

2.1 Sparse Bayesian (Latent) Factor Analysis

The advantages of *Bayesian (latent) factor analysis model* [14] over likelihood (latent) factor analysis model are that

1. The knowledge regarding the model parameters from experts and previous investigations can be incorporated through the prior.

2. The values of parameters are refined using the current training observations.

The Bayesian factor analysis model [14] can be formulated as

$$(\boldsymbol{b}|\boldsymbol{\mu},\boldsymbol{A},\boldsymbol{x},k) = \boldsymbol{\mu} + \boldsymbol{A}\boldsymbol{x} + \boldsymbol{\varepsilon}, \tag{2.1}$$

where $\boldsymbol{b} \in \mathbb{R}^{m \times 1}$ is an observed multivariate variable, $\boldsymbol{\mu} \in \mathbb{R}^{m \times 1}$ is the population mean, $\boldsymbol{A} \in \mathbb{R}^{m \times k}$ is *latent factor loading matrix*, and $\boldsymbol{x} \in \mathbb{R}^{k \times 1}$ is *latent factor score* ($k \ll m$), and $\boldsymbol{\varepsilon} \in \mathbb{R}^{m \times 1}$ is an idiosyncratic *error* term. In Equation (2.1), we list the model parameters, $\{\boldsymbol{\mu},\boldsymbol{A},\boldsymbol{x},k\}$, explicitly on the righthand side of "—". This model is restricted by the following constraints or assumptions:

1. The error term is normally distributed with mean $\mathbf{0}$ and covariance $\boldsymbol{\Phi}$: $\boldsymbol{\varepsilon} \sim N(\mathbf{0},\boldsymbol{\Phi})$. $\boldsymbol{\Phi}$ is diagonal on average.

2. The factor score vector is also normally distributed with mean $\mathbf{0}$ and identity covariance $\boldsymbol{R} = \boldsymbol{I}$: $\boldsymbol{x} \sim N(\mathbf{0},\boldsymbol{R})$; and the factor loading vector is normally distributed: $\boldsymbol{a}_i \sim N(\mathbf{0},\boldsymbol{\Delta})$ where $\boldsymbol{\Delta}$ is diagonal. Alternatively, the factor loading vectors can be normally distributed with mean $\mathbf{0}$ and identity covariance $\boldsymbol{\Delta} = \boldsymbol{I}$; and the factor score vector is normally distributed with mean $\mathbf{0}$ and diagonal covariance $\boldsymbol{R}$. The benefit of identity covariance either on $\boldsymbol{x}$ or $\boldsymbol{A}$ is that arbitrary scale interchange between $\boldsymbol{A}$ and $\boldsymbol{x}$ due to scale invariance can be avoided.

3. $\boldsymbol{x}$ is independent of $\boldsymbol{\varepsilon}$.

For n training instances $\boldsymbol{D}$, we have the likelihood:

$$p(\boldsymbol{D}|\boldsymbol{\mu},\boldsymbol{A},\boldsymbol{Y},\boldsymbol{\Phi},k) = \frac{1}{(2\pi)^{\frac{mn}{2}}|\boldsymbol{\Phi}|^{\frac{n}{2}}} e^{-\frac{1}{2}\sum_{i=1}^{n}(\boldsymbol{d}_i-\boldsymbol{\mu}-\boldsymbol{A}\boldsymbol{y}_i)^{\mathrm{T}}\boldsymbol{\Phi}^{-1}(\boldsymbol{d}_i-\boldsymbol{\mu}-\boldsymbol{A}\boldsymbol{y}_i)} \tag{2.2}$$

$$= \frac{1}{(2\pi)^{\frac{mn}{2}}|\boldsymbol{\Phi}|^{\frac{n}{2}}} e^{-\frac{1}{2}\mathrm{trace}[(\boldsymbol{D}-\boldsymbol{\mu}\mathbf{1}^{\mathrm{T}}-\boldsymbol{A}\boldsymbol{Y})^{\mathrm{T}}\boldsymbol{\Phi}^{-1}(\boldsymbol{D}-\boldsymbol{\mu}\mathbf{1}^{\mathrm{T}}-\boldsymbol{A}\boldsymbol{Y})]}, \tag{2.3}$$

where $\mathrm{trace}(\boldsymbol{M})$ is the trace of square matrix $\boldsymbol{M}$.

The variants of Bayesian factor analysis models differ in the decomposition of the joint priors. The simplest one may be $p(\boldsymbol{\mu},\boldsymbol{A},\boldsymbol{Y}) = p(\boldsymbol{\mu})p(\boldsymbol{A})p(\boldsymbol{Y})$. Suppose k is fixed a priori. The posterior therefore becomes

$$p(\boldsymbol{\mu},\boldsymbol{A},\boldsymbol{Y}|\boldsymbol{D},k) \propto p(\boldsymbol{D}|\boldsymbol{\mu},\boldsymbol{A},\boldsymbol{Y},\boldsymbol{\Phi},k)p(\boldsymbol{\mu})p(\boldsymbol{A})p(\boldsymbol{Y}). \tag{2.4}$$

The model parameters are usually estimated via *Markov chain Monte Carlo* (MCMC) techniques.

Sparse Bayesian factor analysis model imposes a sparsity-inducing distribution over the factor loading matrix instead of Gaussian distribution. In [15], the following mixture of prior is proposed:

$$p(a_{ij}) = (1-\pi_{ij})\delta_0(a_{ij}) + \pi_{ij}N(a_{ij}|0,1), \tag{2.5}$$

where π_{ij} is the probability of a nonzero a_{ij} and $\delta_0(\cdot)$ is the Dirac delta function at 0. Meanwhile, $\boldsymbol{A}$ is constrained using the lower triangular method. *Bayesian factor regression model* (BFRM) is the combination of Bayesian factor analysis and Bayesian regression [15]. It has been applied in oncogenic pathway studies [8] as a variable selection method.

2.2 Bayesian Sparse Representation

Sparse representation (SR) is a principle that a signal can be approximated by a sparse linear combination of dictionary atoms [16]. The SR model can be formulated as

$$\begin{aligned}(\boldsymbol{b}|\boldsymbol{A},\boldsymbol{x},k) &= x_1\boldsymbol{a}_1 + \cdots + x_k\boldsymbol{a}_k + \boldsymbol{\varepsilon} \\ &= \boldsymbol{A}\boldsymbol{x} + \boldsymbol{\varepsilon},\end{aligned} \tag{2.6}$$

where $\boldsymbol{A} = [\boldsymbol{a}_1,\cdots,\boldsymbol{a}_k]$ is called *dictionary*, $\boldsymbol{a}_i$ is a dictionary atom, $\boldsymbol{x}$ is a sparse *coefficient vector*, and $\boldsymbol{\varepsilon}$ is an error term. $\boldsymbol{A}$, $\boldsymbol{x}$, and k are the model parameters. SR model has the following constraints:

1. The error term is Gaussian distributed with mean zero and isotropic covariance, that is $\boldsymbol{\varepsilon} \sim N(\mathbf{0},\boldsymbol{\Phi})$ where $\boldsymbol{\Phi} = \phi\boldsymbol{I}$ where ϕ is a positive scalar.

2. The dictionary atoms is usually Gaussian distributed, that is $\boldsymbol{a}_i \sim N(\mathbf{0},\boldsymbol{\Delta})$ where $\boldsymbol{\Delta} = \boldsymbol{I}$. The coefficient vector should follows a sparsity-inducing distribution.

3. $\boldsymbol{x}$ is independent of $\boldsymbol{\varepsilon}$.

Through comparing the concepts of Bayesian factor analysis and Bayesian sparse representation, we can find that the main difference between them is that the former applies a sparsity-inducing distribution over the factor loading matrix, while the latter uses a sparsity-inducing distribution on the factor score vector.

Sparse representation involves sparse coding and dictionary learning. Given a new signal $\boldsymbol{b}$ and a dictionary $\boldsymbol{A}$, learning the sparse coefficient $\boldsymbol{x}$ is termed *sparse coding*. It can be statistically formulated as

$$(\boldsymbol{b}|\boldsymbol{A}) = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{\varepsilon}. \tag{2.7}$$

Suppose the coefficient vector has Laplacian prior with zero mean and isotropic variance, that is $p(\boldsymbol{x}|\boldsymbol{\Gamma}) = L(\boldsymbol{0},\boldsymbol{\Gamma}) = \frac{1}{(2\gamma)^k} e^{-\frac{\|\boldsymbol{x}\|_1}{\gamma}}$. The likelihood is Gaussian distributed as $p(\boldsymbol{b}|\boldsymbol{A},\boldsymbol{x},\boldsymbol{\Phi}) = N(\boldsymbol{A}\boldsymbol{x},\boldsymbol{\Phi}) = \frac{1}{(2\pi)^{\frac{m}{2}}\phi^{\frac{m}{2}}} e^{-\frac{1}{2\phi}\|\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}\|_2^2}$. The posterior is thus

$$\begin{aligned} p(\boldsymbol{x}|\boldsymbol{A},\boldsymbol{b},\boldsymbol{\Phi},\boldsymbol{\Gamma}) &= \frac{p(\boldsymbol{b}|\boldsymbol{A},\boldsymbol{x},\boldsymbol{\Phi},\boldsymbol{\Gamma})p(\boldsymbol{x}|\boldsymbol{A},\boldsymbol{\Phi},\boldsymbol{\Gamma})}{p(\boldsymbol{b})} \\ &\propto p(\boldsymbol{b}|\boldsymbol{A},\boldsymbol{x},\boldsymbol{\Phi})p(\boldsymbol{x}|\boldsymbol{\Gamma}). \end{aligned} \tag{2.8}$$

Taking the logarithm, the above is thus

$$\begin{aligned} L(\boldsymbol{x}) &= \log p(\boldsymbol{b}|\boldsymbol{A},\boldsymbol{x}) + \log p(\boldsymbol{x}) \\ &= -\frac{1}{2\phi}\|\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}\|_2^2 - \frac{\|\boldsymbol{x}\|_1}{\gamma} + c, \end{aligned} \tag{2.9}$$

where c is a constant term. We can see that maximizing the posterior is equivalent to minimizing the following task:

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}\|_2^2 + \lambda\|\boldsymbol{x}\|_1, \tag{2.10}$$

where $\lambda = \frac{\phi}{\gamma}$. Hereafter we call Equation (2.10) *l_1-least squares* (l_1LS) sparse coding model. It is known as the l_1-regularized regression model in regularization theory. It coincides with the well-known *LASSO* model [17], which in fact is a *maximum a posteriori* (MAP) estimation.

Given training data $\boldsymbol{D}$, learning (or estimating) the dictionary $\boldsymbol{A}$, the coefficient vectors $\boldsymbol{Y}$, and the number of dictionary atoms k is called *dictionary learning*. Suppose k is given a priori, and consider the Laplacian prior over $\boldsymbol{Y}$ and the Gaussian prior over $\boldsymbol{A}$, and suppose $p(\boldsymbol{A},\boldsymbol{Y}) = p(\boldsymbol{A})p(\boldsymbol{Y}) = \prod_{i=1}^{k}(p(\boldsymbol{a}_i))\prod_{i=1}^{n}(p(\boldsymbol{y}_i))$. We thus have the prior:

$$p(\boldsymbol{A},\boldsymbol{Y}|\boldsymbol{\Delta},\boldsymbol{\Gamma}) = \frac{1}{(2\pi)^{\frac{k}{2}}} e^{\sum_{i=1}^{k} -\frac{1}{2}\|\boldsymbol{a}_i\|_2^2} \frac{1}{(2\gamma)^{kn}} e^{\sum_{i=1}^{n} -\frac{\|\boldsymbol{y}_i\|_1}{\gamma}} \tag{2.11}$$

The likelihood is

$$p(\boldsymbol{D}|\boldsymbol{A},\boldsymbol{Y},\boldsymbol{\Phi}) = \frac{1}{(2\pi)^{\frac{mn}{2}}\phi^{\frac{mn}{2}}} e^{-\frac{1}{2\phi}\mathrm{trace}(\|\boldsymbol{D}-\boldsymbol{A}\boldsymbol{Y}\|_F^2)}. \tag{2.12}$$

The posterior is

$$p(\boldsymbol{A},\boldsymbol{Y}|\boldsymbol{D},\boldsymbol{\Delta},\boldsymbol{\Gamma},\boldsymbol{\Phi}) = \frac{p(\boldsymbol{D}|\boldsymbol{A},\boldsymbol{Y},\boldsymbol{\Delta},\boldsymbol{\Gamma},\boldsymbol{\Phi})p(\boldsymbol{A},\boldsymbol{Y}|\boldsymbol{\Delta},\boldsymbol{\Gamma},\boldsymbol{\Phi})}{p(\boldsymbol{D})} \tag{2.13}$$

$$\propto p(\boldsymbol{D}|\boldsymbol{A},\boldsymbol{Y},\boldsymbol{\Phi})p(\boldsymbol{A}|\boldsymbol{\Delta})p(\boldsymbol{Y}|\boldsymbol{\Gamma}). \tag{2.14}$$

Ignoring the normalization term, the log-posterior is thus

$$L(\boldsymbol{A},\boldsymbol{Y}) = -\sum_{i=1}^{n}\frac{1}{2\phi}\|\boldsymbol{d}_i-\boldsymbol{A}\boldsymbol{y}_i\|_2^2-\sum_{i=1}^{k}\frac{1}{2}\|\boldsymbol{a}_i\|_2^2-\sum_{i=1}^{n}\frac{\|\boldsymbol{y}_i\|_1}{\gamma}+c. \tag{2.15}$$

Therefore the MAP estimation of dictionary learning task is

$$\min_{\boldsymbol{A},\boldsymbol{Y}} f(\boldsymbol{A},\boldsymbol{Y}) = \sum_{i=1}^{n}\frac{1}{2}\|\boldsymbol{d}_i-\boldsymbol{A}\boldsymbol{y}_i\|_2^2+\sum_{i=1}^{k}\frac{\alpha}{2}\|\boldsymbol{a}_i\|_2^2+\sum_{i=1}^{n}\lambda\|\boldsymbol{y}_i\|_1, \tag{2.16}$$

where $\alpha=\phi$ and $\lambda=\frac{\phi}{\gamma}$. Equation (2.16) is known as a dictionary learning model based on l_1-regularized least squares.

3 Bayesian Sparse Coding

Since, the l_1LS sparse coding (Equation (2.10)) is a two-sided symmetric model, thus a coefficient can be zero, positive, or negative [18]. In Bioinformatics, l_1LS sparse coding has been applied for the classification of microarray gene expression data in [19]. The main idea is in the following. First, training instances are collected in a dictionary. Then, a new instance is regressed by l_1LS sparse coding. Thus its corresponding sparse coefficient vector is obtained. Next, the regression residual of this instance to each class is computed, and finally this instance is assigned to the class with the minimum residual.

We generalize this methodology in the way that the sparse codes can be obtained by many other regularization methods and constraints. For example, we can pool all training instances in a dictionary (hence $k=n$ and $\boldsymbol{A}=\boldsymbol{D}$), and then learn the non-negative coefficient vectors of a new instance, which is formulated as an one-sided model:

$$\min_{\boldsymbol{x}}\frac{1}{2}\|\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}\|_2^2 \text{ s.t. } \boldsymbol{x}\geq 0. \tag{2.17}$$

We called this model the *non-negative least squares* (NNLS) sparse coding. NNLS has two advantages over l_1LS. First, the non-negative coefficient vector is more easily interpretable than coefficient vector of mixed signs, under some circumstances. Second, NNLS is a non-parametric model which is more convenient in practice. From a Bayesian viewpoint, Equation (2.17) is equivalent to the MAP estimation with the same

Gaussian error as in Equation (2.6), but with the following discrete prior:

$$p(\boldsymbol{x}) = \begin{cases} 0.5^k & \text{if } \boldsymbol{x} \geq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{2.18}$$

This non-negative prior implies that, the elements in $\boldsymbol{x}$ are independent, and the probability that $x_i = 0$ is 0.5 and the probability that $x_i > 0$ is 0.5 as well. (That is the probabilities of x_i being either 0 or positive are equal, and the probability of being negative is zero.) Inspired by many sparse NMFs, l_1-regularization can be additionally used to produce more sparse coefficients than NNLS above. The combination of l_1-regularization and non-negativity constraint results in the l_1NNLS sparse coding model as formulated below:

$$\min_{\boldsymbol{x}} \frac{1}{2}\|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\|_2^2 + \boldsymbol{\lambda}^{\mathrm{T}}\boldsymbol{x} \text{ s.t. } \boldsymbol{x} \geq 0. \tag{2.19}$$

We name Equation (2.19) the l_1*NNLS* model. It is more flexible than NNLS, because it can produce more sparse coefficients as controlled by λ. This model in fact uses the following prior:

$$p(\boldsymbol{x}) = \begin{cases} \frac{1}{\gamma^k} e^{-\frac{\|\boldsymbol{x}\|_1}{\gamma}} & \text{if } \boldsymbol{x} \geq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{2.20}$$

Now, we give the generalized sparse-coding-based classification approach in details. The method is depicted in Algorithm 1. We shall later discuss the optimization algorithms, required in the first step. The NN rule mentioned in Algorithm 1 is inspired by the usual way of using NMF as a clustering method. Suppose there are C classes with labels $1, \cdots, C$. For a given new instance $\boldsymbol{b}$, its class is $l = \arg\max_{i=1,\cdots,k} x_i$ (where $k = n$ is the number of training samples). It selects the maximum coefficient in the coefficient vector, say x_l, and then assigns the class label of the corresponding training instance, say c_l, to this new instance. Essentially, this rule is equivalent to applying *nearest neighbor* (NN) classifier in the column space of the training instances. In this space, the representations of the training instances are identity matrix. The NN rule can be further generalized to the weighted *K-NN* rule. Suppose a K-length vector $\bar{\boldsymbol{x}}$ accommodates the K-largest coefficients from $\boldsymbol{x}$, and $\bar{\boldsymbol{c}}$ has the corresponding K class labels. The class label of $\boldsymbol{b}$ can be designated as $l = \arg\max_{i=1,\cdots,C} s_i$ where $s_i = \mathrm{sum}(\delta_i(\bar{\boldsymbol{x}}))$. $\delta_i(\bar{\boldsymbol{x}})$ is a K-length vector and is defined as

$$(\delta_i(\bar{\boldsymbol{x}}))_j = \begin{cases} \bar{x}_j & \text{if } \bar{c}_j = i, \\ 0 & \text{otherwise .} \end{cases} \tag{2.21}$$

The maximum value of K can be k, the number of dictionary atoms. In this case, K is in fact the number of all non-zeros in $\boldsymbol{x}$. Alternatively, the *nearest subspace* (NS) rule, proposed in [20], can be used to interpret the sparse coding. NS rule has the advantage of the discrimination property in the sparse coefficients. It assigns the class with the minimum regression residual to $\boldsymbol{b}$. Mathematically, it is expressed as $j = \min_{1 \leq i \leq C} r_i(\boldsymbol{b})$ where $r_i(\boldsymbol{b})$ is the regression residual corresponding to the i-th class and is computed as $r_i(\boldsymbol{b}) = \|\boldsymbol{b} - \boldsymbol{A}\delta_i(\boldsymbol{x})\|_2^2$,

where $\boldsymbol{\delta}_i(\boldsymbol{x})$ is defined analogously as in Equation (2.21).

Algorithm 1 *Sparse-Coding Based Classification*

Input: $\boldsymbol{A}_{m\times n}$: n training instances, $\boldsymbol{c}$: class labels, $\boldsymbol{B}_{m\times p}$: p new instances
Output: $\boldsymbol{p}$: predicted class labels of the p new instances

1. Normalize each instance to have unit l_2-norm;
2. Learn the sparse coefficient matrix $\boldsymbol{X}$, of the new instances by solving Equation (2.10), (2.17), or (2.19);
3. Use a sparse interpreter to predict the class labels of new instances, e.g. the NN, K-NN, or NS rule;

4 Optimization for Sparse Coding

In this section, we focus on the optimizations for the models of sparse coding. We first show that the optimizations are essentially quadratic programmes and dimension-free. We then review the existing methods including interior-point method and proximal method. After that, we propose our active-set methods and decomposition methods. Finally, we state that the all these methods can be extended to their kernel versions.

4.1 The Optimizations are Quadratic Programmes and Dimension-Free

The problem in Equation (2.10) is equivalent to the following unconstrained non-smooth (that is undifferentiable at some points) *quadratic programming* (QP):

$$\min_{\boldsymbol{x}} \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x}+\boldsymbol{g}^{\mathrm{T}}\boldsymbol{x}+\lambda\|\boldsymbol{x}\|_1, \tag{2.22}$$

where $\boldsymbol{H}_{k\times k}=\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$, and $\boldsymbol{g}=-\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}$. We thus know that the l_1LS problem is a l_1QP problem. This can be converted to the following smooth (that is differentiable everywhere) constrained QP problem:

$$\min_{\boldsymbol{x},\boldsymbol{u}} \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x}+\boldsymbol{g}^{\mathrm{T}}\boldsymbol{x}+\boldsymbol{\lambda}^{\mathrm{T}}\boldsymbol{u} \text{ s.t. } -\boldsymbol{u}\le\boldsymbol{x}\le\boldsymbol{u}, \tag{2.23}$$

where $\boldsymbol{u}$ is an auxiliary vector variable to squeeze $\boldsymbol{x}$ towards zero. It can be further written into the standard form:

$$\begin{aligned}\min_{\boldsymbol{x},\boldsymbol{u}} &\frac{1}{2}[\boldsymbol{x}^{\mathrm{T}},\boldsymbol{u}^{\mathrm{T}}]\begin{bmatrix}\boldsymbol{H} & \boldsymbol{0}_{k\times k}\\ \boldsymbol{0}_{k\times k} & \boldsymbol{0}_{k\times k}\end{bmatrix}\begin{bmatrix}\boldsymbol{x}\\ \boldsymbol{u}\end{bmatrix}+\boldsymbol{g}^{\mathrm{T}}\boldsymbol{x}+\boldsymbol{\lambda}^{\mathrm{T}}\boldsymbol{u} \\ s.t. &\begin{bmatrix}\boldsymbol{I}_{k\times k} & -\boldsymbol{I}_{k\times k}\\ -\boldsymbol{I}_{k\times k} & -\boldsymbol{I}_{k\times k}\end{bmatrix}\le 0,\end{aligned} \tag{2.24}$$

where $\boldsymbol{0}_{k\times k}$ is a null square matrix, $\boldsymbol{I}_{k\times k}$ is an identity matrix. Obviously, the Hessian in this problem is positive semidefinite as we always suppose $\boldsymbol{H}$ is positive semidefinite in this chapter. A symmetric squares

matrix $\boldsymbol{H}_{k\times k}$ is said to be positive semidefinite, if and only if $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}_{k\times k}\boldsymbol{x} \geq 0$, $\forall \boldsymbol{x} \in \mathbb{R}^k$.

Both the NNLS problem in Equation (2.17) and the l_1NNLS problem in Equation (2.19) can be easily reformulated to the following *non-negative QP* (NNQP) problem:

$$\min \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x} + \boldsymbol{g}^{\mathrm{T}}\boldsymbol{x} \text{ s.t. } \boldsymbol{x} \geq 0, \tag{2.25}$$

where $\boldsymbol{H} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$, $\boldsymbol{g} = -\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}$ for NNLS, and $\boldsymbol{g} = -\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b} + \boldsymbol{\lambda}$ for l_1NNLS.

4.2 Interior-Point Method

The log-barrier interior-point method for the l_1LS problem (Equation (2.10)) is proposed in [21]. The basic idea is to convert non-smooth problem into unconstrained problem. For the convenience of discussion, we extend this method to solve the l_1QP and NNQP problems. The interior-point method for l_1QP is introduced in the following. Due to similarity of derivation, we omit the interior-point method for NNQP problem. The problem in Equation (2.23) can be transformed into minimizing the unconstrained log-barrier function:

$$f_t(\boldsymbol{x},\boldsymbol{u}) = t(\frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x} + \boldsymbol{g}^{\mathrm{T}}\boldsymbol{x} + \boldsymbol{\lambda}^{\mathrm{T}}\boldsymbol{u}) - \sum_{i=1}^{k}\log(u_i^2 - x_i^2), \tag{2.26}$$

where $-\sum_{i=1}^{k}\log(u_i^2 - x_i^2)$ is the barrier penalty function, and t is positive parameter to control the penalty.

We can obtain its gradient as below

$$\boldsymbol{d} = \begin{bmatrix} (\frac{\partial f_t(\boldsymbol{x},\boldsymbol{u})}{\partial \boldsymbol{x}})^T \\ (\frac{\partial f_t(\boldsymbol{x},\boldsymbol{u})}{\partial \boldsymbol{u}})^T \end{bmatrix} = \begin{bmatrix} t(\boldsymbol{H}\boldsymbol{x} + \boldsymbol{g}) + 2[\frac{x_1}{u_1^2 - x_1^2} \cdots \frac{x_k}{u_k^2 - x_k^2}]^{\mathrm{T}} \\ t\boldsymbol{\lambda} - 2[\frac{u_1}{u_1^2 - x_1^2} \cdots \frac{u_k}{u_k^2 - x_k^2}]^{\mathrm{T}} \end{bmatrix}. \tag{2.27}$$

And its Hessian matrix is

$$\boldsymbol{H} = \begin{bmatrix} \frac{\partial f_t^2(\boldsymbol{x},\boldsymbol{u})}{\partial \boldsymbol{x}^2} & \frac{\partial f_t^2(\boldsymbol{x},\boldsymbol{u})}{\partial \boldsymbol{u}\partial \boldsymbol{x}} \\ \frac{\partial f_t^2(\boldsymbol{x},\boldsymbol{u})}{\partial \boldsymbol{x}\partial \boldsymbol{u}} & \frac{\partial f_t^2(\boldsymbol{x},\boldsymbol{u})}{\partial \boldsymbol{u}^2} \end{bmatrix}, \tag{2.28}$$

Where

$$\frac{\partial f_t^2(\boldsymbol{x},\boldsymbol{u})}{\partial \boldsymbol{x}^2} = t\boldsymbol{H} + 2\mathrm{diag}(\frac{u_1^2 + x_1^2}{(u_1^2 - x_1^2)^2} \cdots \frac{u_k^2 + x_k^2}{(u_k^2 - x_k^2)^2}), \tag{2.29}$$

$$\frac{\partial f_t^2(\boldsymbol{x},\boldsymbol{u})}{\partial \boldsymbol{u}^2} = 2\mathrm{diag}(\frac{u_1^2 + x_1^2}{(u_1^2 - x_1^2)^2} \cdots \frac{u_k^2 + x_k^2}{(u_k^2 - x_k^2)^2}), \tag{2.30}$$

and

$$\frac{\partial f_t^2(\boldsymbol{x},\boldsymbol{u})}{\partial \boldsymbol{x}\partial \boldsymbol{u}} = -4\mathrm{diag}(\frac{x_1 u_1}{(u_1^2 - x_1^2)^2} \cdots \frac{x_k u_k}{(u_k^2 - x_k^2)^2}). \tag{2.31}$$

The optimization of the log-barrier method is to solve a sequence of $f_t(\boldsymbol{x},\boldsymbol{u})$ (Equation (2.26)), as t increases. Newton's method can be used to minimize the unconstrained problem in Equation (2.26), given a positive t [22]. As t increases to positive infinite, the solution converges to the optimal solution. Given a value of t, the convergence rate of Newton's method is super-linear or quadratic with heavy steps.

4.3 Proximal Method

As one of fast first-order methods, the proximal method was proposed in [23] to solve the l_1LS problem. Compared with the interior-point method, it has two advantages. First, the Hessian is not recomputed in each step. Second, an analytical solution can be obtained in each step. As in the interior-point method, we extend the proximal method for the l_1QP problem for the convenience of discussion. The main idea is in the following. The proximal method works in an iterative procedure. Given the current estimate $\hat{\boldsymbol{x}}$, the new estimate is the solution of the following first-order Taylor series of Equation (2.22):

$$\min_{\boldsymbol{x}} f(\hat{\boldsymbol{x}}) + \nabla q(\hat{\boldsymbol{x}})(\boldsymbol{x}-\hat{\boldsymbol{x}}) + \lambda\|\boldsymbol{x}\|_1 + \frac{L}{2}\|\boldsymbol{x}-\hat{\boldsymbol{x}}\|_2^2, \tag{2.32}$$

where $q(\hat{\boldsymbol{x}}) = \frac{1}{2}\hat{\boldsymbol{x}}^{\mathrm{T}}\boldsymbol{H}\hat{\boldsymbol{x}} + \boldsymbol{g}^{\mathrm{T}}\hat{\boldsymbol{x}}$ and $L > 0$ is a parameter. This is done iteratively until termination criteria are met. Equation (2.32) is equivalent to the following proximal operator:

$$\min_{\boldsymbol{x}} \frac{1}{2}\|\boldsymbol{x}-\bar{\boldsymbol{x}}\|_2^2 + \eta\|\boldsymbol{x}\|_1, \tag{2.33}$$

where $\bar{\boldsymbol{x}} = \hat{\boldsymbol{x}} - \frac{1}{L}\nabla q(\hat{\boldsymbol{x}})$, and $\eta = \frac{\lambda}{L}$. Unlike the interior-point method with *heavy* Newton step in each iteration where the Hessian is involved in the update, it has element-wise analytical solution:

$$x_i = \begin{cases} \mathrm{sign}(\bar{x}_i)(|\bar{x}_i| - \eta) & \text{if } |\bar{x}_i| > \eta \\ 0 & \text{otherwise} \end{cases}. \tag{2.34}$$

The proximal method for l_1QP has *linear* convergence rate with *swift* steps where gradient is only involved.

4.4 Active-Set Algorithm for l_1LS

A general active-set algorithm for constrained QP is provided in [24], where the main idea is that a working set is updated iteratively until it meets the true active set. In each iteration, a new solution $\boldsymbol{x}_t$ to the QP constrained only by the current working set is obtained. If the update step $\boldsymbol{p}_t = \boldsymbol{x}_t - \boldsymbol{x}_{t-1}$ is zero, then Lagrangian multipliers of the current active inequalities are computed. If all these multipliers corresponding to the working set are non-negative, then the algorithm terminates with an optimal solution. Otherwise, an active inequality is dropped from the current working set. If the update step $\boldsymbol{p}_t$ is nonzero, then an update length α is computed using the inequality of the current passive set. The new solution is updated as $\boldsymbol{x}_t = \boldsymbol{x}_{t-1} + \alpha\boldsymbol{p}_t$. If $\alpha < 1$, then a blocking inequality is added to the working set.

To solve our specific problem efficiently in Equation (2.24), we have to modify the general method, because i) our constraint is sparse, and for the i-th constraint, we have $x_i - u_i \leq 0$ (if $i \leq k$) or $-x_i - u_i \leq 0$ (if $i \geq k+1$); and ii) when u_i is not constrained in the current working set, the QP constrained by the working set is unbounded, therefore it is not necessary to solve this problem to obtain $\boldsymbol{p}_t$. In the latter situation, $\boldsymbol{p}_t$ is unbounded. This could cause some issues in numerical computation. Solving the unbounded problem is time-consuming if the algorithm is unaware of the unbounded issue. If $\boldsymbol{p}_t$ contains values equal to $\pm\infty$, then

the algorithm may crash.

In Algorithm 2, we revise the active-set algorithm for l_1LS sparse coding. To address the potential issues above, we have the following four modifications. First, we require that the working set is *complete*. That is all the variables in $\boldsymbol{u}$ must be constrained when computing the current update step. And therefore all variables in $\boldsymbol{x}$ are also constrained due to the specific structure of the constraints in our problem. For example, if $k = 3$, a working set $\{1,2,6\}$ is complete as all variables, $x_1, x_2, x_3, u_1, u_2, u_3$, are constrained, while $\{1,2,4\}$ is not complete, as u_3 (and x_3) is not constrained. Second, the update step of the variables that are constrained once in the working set are computed by solving the equality constrained QP. The variables constrained twice are directly set to zeros. In the example above, suppose the current working set is $\{1,2,4,6\}$, then x_2, x_3, u_2, u_3 are computed by the constrained QP, while x_1 and u_1 are zeros. This is because the only value satisfying the constraint $-u_1 = x_1 = u_1$ is $x_1 = u_1 = 0$. Third, in this example, we do not need to solve the equality constrained QP with four variables. In fact we only need two variables by setting $u_2 = -x_2$ and $u_3 = x_3$. Fourth, once a constraint is dropped from the working set and it becomes incomplete, other inequalities must be immediately added to it until it is complete. In the initialization of Algorithm 2, we can alternatively initialize $\boldsymbol{x}$ by 0's. This is more efficient than initializing $\boldsymbol{x} = (\boldsymbol{H})^{-1}(-\boldsymbol{g})$ for large-scale and very sparse problems.

4.5 Active-Set Algorithm for NNLS and l_1NNLS

Now, we present the active-set algorithm for NNQP. This problem is easier to solve than l_1QP as the scale of the Hessian of NNQP is half that of l_1QP and the constraint is much simpler. Our algorithm is obtained through generalizing the famous active-set algorithm for NNLS originally by [25]. The generalized algorithm is given in Algorithm 3. The warm-start point is initialized by the solution to the unconstrained QP. As in Algorithm 2, $\boldsymbol{x}$ can be alternatively initialized by 0's. The algorithm keeps adding and dropping constraints in the working set until the true active set is found.

4.6 Parallel Active-Set Algorithms

The formulations of l_1QP and NNQP sparse coding for p new instances are, respectively,

$$\begin{aligned} &\min_{\boldsymbol{X},\boldsymbol{U}} \sum_{i=1}^{p} \frac{1}{2}\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x}_i + \boldsymbol{g}_i^{\mathrm{T}}\boldsymbol{x}_i + \boldsymbol{\lambda}^{\mathrm{T}}\boldsymbol{u}_i, \\ &\text{s.t. } -\boldsymbol{U} \leq \boldsymbol{X} \leq \boldsymbol{U}, \end{aligned} \tag{2.37}$$

and

$$\min_{\boldsymbol{X}} \sum_{i=1}^{p} \frac{1}{2}\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x}_i + \boldsymbol{g}_i^{\mathrm{T}}\boldsymbol{x}_i \text{ s.t. } \boldsymbol{X} \geq 0. \tag{2.38}$$

If we want to classify multiple new instances using the classification method described in Algorithm 1, the initial idea in [20] and [19] is to optimize the sparse coding one at a time. The interior-point algorithm,

Algorithm 2 *Active-Set l_1QP Algorithm*

Input: Hessian $\boldsymbol{H}_{k\times k}$, vector $\boldsymbol{g}_{k\times 1}$, scalar λ

Output: vector $\boldsymbol{x}$ which is a solution to $\min \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x}+\boldsymbol{g}^{\mathrm{T}}\boldsymbol{x}+\boldsymbol{\lambda}^{\mathrm{T}}\boldsymbol{u}$, s.t. $-\boldsymbol{u}\leq\boldsymbol{x}\leq\boldsymbol{u}$

{initialize the algorithm by a feasible solution and a complete working set}
$\boldsymbol{x}=(\boldsymbol{H})^{-1}(-\boldsymbol{g})$; $\boldsymbol{u}=|\boldsymbol{x}|$;
$\mathcal{R}=\{i,j|\forall i:$ if $x_i>0$ let $j=k+i$ otherwise $j=i\}$; {initialize working set}
$\mathcal{P}=\{1:2k\}-\mathcal{R}$; {initialize inactive(passive) set}

while true **do**
{compute update step}
Let $\mathcal{R}_{\text{once}}$ be the indices of variables constrained once by $\mathcal{R}$;
$\boldsymbol{p}_{2k\times 1}=0$;

$$\boldsymbol{p}_{\boldsymbol{x},\mathcal{R}_{\text{once}}}=\arg\min_{\boldsymbol{q}}\boldsymbol{q}^{\mathrm{T}}\boldsymbol{H}_{\mathcal{R}_{\text{once}}}\boldsymbol{q}+[\boldsymbol{H}_{\mathcal{R}_{\text{once}}}\boldsymbol{x}_{\mathcal{R}_{\text{once}}}+\boldsymbol{g}_{\mathcal{R}_{\text{once}}}+\lambda\boldsymbol{e}]^{\mathrm{T}}\boldsymbol{q}, \tag{2.35}$$

where $e_i=1$ if $u_{\mathcal{R}_{\text{once}},i}=x_{\mathcal{R}_{\text{once}},i}$, or -1 if $u_{\mathcal{R}_{\text{once}},i}=-x_{\mathcal{R}_{\text{once}},i}$;

if $\boldsymbol{p}=0$ **then**
Obtain Lagrange multiplier $\boldsymbol{\mu}$ by solving

$$\boldsymbol{A}_{\mathcal{R}}^{\mathrm{T}}\boldsymbol{\mu}=-\begin{bmatrix}\boldsymbol{H}\boldsymbol{x}+\boldsymbol{g}\\ \boldsymbol{\lambda}\end{bmatrix}, \tag{2.36}$$

where $\boldsymbol{A}$ is the constraint matrix in Equation (2.24)
if $\mu_i\geq 0\ \forall i\in\mathcal{R}$ **then**
Terminate successfully;
else
$\mathcal{R}=\mathcal{R}-j$; $\mathcal{P}=\mathcal{P}+j$ where $j=\arg\min_{l\in\mathcal{R}}\mu_l$;
Add other passive constraints to $\mathcal{R}$ until it is complete;
end if
end if

if $\boldsymbol{p}\neq 0$ **then**
$\alpha=\min\left(1,\min_{i\in\mathcal{P},\boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{p}\geq 0}\frac{-\boldsymbol{a}_i^{\mathrm{T}}[\boldsymbol{x};\boldsymbol{u}]}{\boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{p}}\right)$;
$[\boldsymbol{x};\boldsymbol{u}]=[\boldsymbol{x};\boldsymbol{u}]+\alpha\boldsymbol{p}$;
if $\alpha<1$ **then**
$\mathcal{R}=\mathcal{R}+i$; $\mathcal{P}=\mathcal{P}-i$, where i corresponds to α;
end if
end if
end while

proposed in [21], is a fast large-scale sparse coding algorithm, and the proximal algorithm in [23] is a fast first-order method whose advantages have been recently highlighted for non-smooth problems. If we adapt both algorithms to solve our multiple l_1QP in Equation (2.37) and NNQP in Equation (2.38), it will be difficult to solve the single problems in parallel and share the computations. Therefore, the time-complexity

Algorithm 3 *Active-Set NNQP Algorithm*

Input: Hessian $\boldsymbol{H}_{k\times k}$, vector $\boldsymbol{g}_{k\times 1}$
Output: vector $\boldsymbol{x}$ which is a solution to $\min \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x}+\boldsymbol{g}^{\mathrm{T}}\boldsymbol{x}$, s.t. $\boldsymbol{x}\geq 0$

$\boldsymbol{x}=[(\boldsymbol{H})^{-1}(-\boldsymbol{g})]_+$; { $\boldsymbol{x}=[\boldsymbol{y}]_+$ is defined as $x_i=y_i$ if $y_i>0$, otherwise $x_i=0$}
$\mathcal{R}=\{i|x_i=0\}$; {initialize active set}
$\mathcal{P}=\{i|x_i>0\}$; {initialize inactive(passive) set}
$\boldsymbol{\mu}=\boldsymbol{H}\boldsymbol{x}+\boldsymbol{g}$; {the lagrange multiplier}

while $\boldsymbol{R}\neq\emptyset$ and $\min_{i\in\mathcal{R}}(\mu_i)<-\varepsilon$ **do**
  {ε is a small positive numerical tolerance}
  $j=\arg\min_{i\in\mathcal{R}}(\mu_i)$; {get the minimal negative multiplier}
  $\mathcal{P}=\mathcal{P}+\{j\}$; $\mathcal{R}=\mathcal{R}-\{j\}$;
  $\boldsymbol{t}_{\mathcal{P}}=(\boldsymbol{H}_{\mathcal{P}})^{-1}(-\boldsymbol{g}_{\mathcal{P}})$;
  $\boldsymbol{t}_{\mathcal{R}}=\boldsymbol{0}$;

  while $\min\boldsymbol{t}_{\mathcal{P}}\leq 0$ **do**
    $\alpha=\min_{i\in\mathcal{P},t_i\leq 0}\frac{x_i}{x_i-t_i}$;
    $\mathcal{K}=\arg\min_{i\in\mathcal{P},t_i\leq 0}\frac{x_i}{x_i-t_i}$; {there is one or several indices correspond to α}
    $\boldsymbol{x}=\boldsymbol{x}+\alpha(\boldsymbol{t}-\boldsymbol{x})$;
    $\mathcal{P}=\mathcal{P}-\mathcal{K}$; $\mathcal{R}=\mathcal{R}+\mathcal{K}$;
    $\boldsymbol{t}_{\mathcal{P}}=(\boldsymbol{H}_{\mathcal{P}})^{-1}(-\boldsymbol{g}_{\mathcal{P}})$;
    $\boldsymbol{t}_{\mathcal{R}}=\boldsymbol{0}$;
  end while

  $\boldsymbol{x}=\boldsymbol{t}$;
  $\boldsymbol{\mu}=\boldsymbol{H}\boldsymbol{x}+\boldsymbol{g}$;
end while

of the multiple problems will be the summation of that of the individual problems. However, the multiple problems can be much more efficiently solved by active-set algorithms. We adapt both Algorithms 2 and 3 to solve multiple l_1QP and NNQP in a parallel fashion. The individual active-set algorithms can be solved in parallel by sharing the computation of matrix inverses (systems of linear equations in essence). At each iteration, single problems having the same active set have the same systems of linear equations to solve. These systems of linear equations can be solved once only. For a large value p, that is large-scale multiple problems, the active-set algorithms have dramatic computational advantage over interior-point [21] and proximal [23] methods unless these methods have a scheme for sharing computations. Additionally, active-set methods are more precise than interior-point methods. Interior-point methods do not allow $u_i^2=x_i^2$ and u_i^2 must be always greater than x_i^2 due to feasibility. But $u_i^2=x_i^2$ is naturally possible when the i-th constraint is active. $u_i=x_i=0$ is reasonable and possible. The active-set algorithms do allow this situation.

4.7 Decomposition Method

A decomposition method [26] has first been devised in the optimization of large-scale SVM which is a QP problem constrained by equality and bound constraints. The basic idea of the decomposition method is, in fact, an implementation of the block-coordinate-descent scheme [27]. The decomposition method works in an iterative procedure. In each iteration, a few number of variables violating the optimality conditions (e.g. *Karush-Kuhn-Tucker* (KKT) conditions) are included in a working set, while the rest are fixed. Only the variables in the working set are updated by a solver. This procedure iterates until no coefficient violates the KKT conditions. Because the objective is decreased in each iteration, the convergence is guaranteed. *Sequential minimal optimization* (SMO) [28] is the extreme case of the decomposition method for SVM, where only a minimal number of variables (two variables) are updated. One of the features of the SMO method for SVM is that the subproblem with only two variables can be solved analytically.

In this section, we propose decomposition methods for the l_1LS and NNLS sparse coding models. Our contributions are as follows

1. To the best of our knowledge, it is the first time that the idea of decomposition is investigated in sparse coding. The decomposition method has been successfully applied in the optimization of support vector machine (SVM).

2. We design sequential minimal optimization (SMO) methods for l_1LS, NNLS, and l_1NNLS sparse coding models, but our methods may be applicable for many other models.

4.8 Decomposition Method for l_1QP

Let $\mathcal{A}$ be the set of a few working variables and $\mathcal{P}$ be the set of fixed variables. The decomposition of $f(\boldsymbol{x})$ in Equation (2.22) can be

$$\begin{aligned} f(\boldsymbol{x}_{\mathcal{A}}) &= \frac{1}{2}[\boldsymbol{x}_{\mathcal{A}}^{\mathrm{T}}, \boldsymbol{x}_{\mathcal{P}}^{\mathrm{T}}] \begin{bmatrix} \boldsymbol{H}_{\mathcal{A}\mathcal{A}} & \boldsymbol{H}_{\mathcal{A}\mathcal{P}} \\ \boldsymbol{H}_{\mathcal{P}\mathcal{A}} & \boldsymbol{H}_{\mathcal{P}\mathcal{P}} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_{\mathcal{A}} \\ \boldsymbol{x}_{\mathcal{P}} \end{bmatrix} + [\boldsymbol{g}_{\mathcal{A}}^{\mathrm{T}}, \boldsymbol{g}_{\mathcal{P}}^{\mathrm{T}}] \begin{bmatrix} \boldsymbol{x}_{\mathcal{A}} \\ \boldsymbol{x}_{\mathcal{P}} \end{bmatrix} + \lambda \| \begin{bmatrix} \boldsymbol{x}_{\mathcal{A}} \\ \boldsymbol{x}_{\mathcal{P}} \end{bmatrix} \|_1 \\ &= \frac{1}{2}\boldsymbol{x}_{\mathcal{A}}^{\mathrm{T}}\boldsymbol{H}_{\mathcal{A}\mathcal{A}}\boldsymbol{x}_{\mathcal{A}} + (\boldsymbol{H}_{\mathcal{A}\mathcal{P}}\boldsymbol{x}_{\mathcal{P}} + \boldsymbol{g}_{\mathcal{A}})^{\mathrm{T}}\boldsymbol{x}_{\mathcal{A}} + \lambda\|\boldsymbol{x}_{\mathcal{A}}\|_1 + constant. \end{aligned} \tag{2.39}$$

We can see that the subproblem corresponding to the working set is a l_1QP problem as well. From this point, a solver of l_1QP is still required. In this study, we focus on the extreme case of Equation (2.39): the SMO method. It can be easily seen that the minimal size of $\mathcal{A}$ is *one* and $\mathcal{P}$ contains the remaining $k-1$ variables. Without loss of generality, we denote such variable by x_1. This results in the following subproblem:

$$\min_{x_1} f(x_1) = \frac{1}{2}h_{11}x_1^2 + b_1x_1 + \lambda|x_1|, \tag{2.40}$$

where $b_1 = \boldsymbol{H}_{1\mathcal{P}}\boldsymbol{x}_{\mathcal{P}} + g_1$.

Analytical Solution

The SMO problem in Equation (2.40) can be solved analytically. Now let us separate the interval into $x_1 \geq 0$ and $x_1 \leq 0$. For $x_1 \geq 0$, the objective $f(x_1)$ becomes

$$f(x_1) = \frac{1}{2}h_{11}x_1^2 + (b_1 + \lambda)x_1. \tag{2.41}$$

Taking first-order derivative and setting it to zero, we have

$$\frac{\partial f(x_1)}{\partial x_1} = h_{11}x_1 + (b_1 + \lambda) = 0. \tag{2.42}$$

We thus have $x_1^{(+)} = \frac{-b_1-\lambda}{h_{11}}$. Therefore, for interval $x_1 \geq 0$, we have the optimal solution:

$$x_1^{(+*)} = \begin{cases} x_1^{(+)} & \text{if } x_1^{(+)} \geq 0 \\ 0 & \text{otherwise} \end{cases}. \tag{2.43}$$

Similarly, for $x_1 \leq 0$, the objective $f(x_1)$ becomes

$$f(x_1) = \frac{1}{2}h_{11}x_1^2 + (b_1 - \lambda)x_1. \tag{2.44}$$

We thus have $x_1^{(-)} = \frac{-b_1+\lambda}{h_{11}}$. And for interval $x_1 \leq 0$, we have the optimal solution:

$$x_1^{(-*)} = \begin{cases} x_1^{(-)} & \text{if } x_1^{(-)} \leq 0 \\ 0 & \text{otherwise} \end{cases}. \tag{2.45}$$

By considering both positive and negative interval together, the optimal solution to Equation (2.40) is the one, among $x_1^{(+*)}$ and $x_1^{(-*)}$, which obtains the minimum objective value, that is $x_1^* = \arg\min_{\{x_1^{(+*)}, x_1^{(-*)}\}} f(x_1)$.

We note that $x_1^{(+)} \geq 0$ is equivalent to $b_1 \leq -\lambda$, and $x_1^{(-)} \leq 0$ is equivalent to $b_1 \geq \lambda$. We can state that if $b_1 \leq -\lambda$ or $b_1 \geq \lambda$ the solution to Equation (2.40) is $x_1^* = \frac{-b_1-\lambda}{h_{11}}$ or $x_1^* = \frac{-b_1+\lambda}{h_{11}}$, respectively. Otherwise, $x_1^* = 0$. Therefore, the solution to Equation (2.40) can be equivalently written as

$$x_1^* = \begin{cases} \frac{-\text{sign}(b_1)(|b_1|-\lambda)}{h_{11}} & \text{if } |b_1| \geq \lambda \\ 0 & \text{otherwise} \end{cases}. \tag{2.46}$$

From this, we can obtain a general proposition below which is very useful:

Proposition 1. *The solution to the following problem*

$$\min_x f(x) = x^2 + bx + \lambda|x| \tag{2.47}$$

is analytically

$$x^* = \begin{cases} -sign(b)(|b| - \lambda) & if\ |b| \geq \lambda \\ 0 & otherwise \end{cases}. \tag{2.48}$$

Now that we have known how to update a working variable. In the following, we show how to select a working variable.

Select x_1 Which Violates the Optimality Condition

The optimality condition of Equation (2.22) is

$$\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H} + \boldsymbol{g}^{\mathrm{T}} + \nabla\lambda\|\boldsymbol{x}\|_1 = 0. \tag{2.49}$$

However, because $\lambda\|\boldsymbol{x}\|_1$ is not differentiable, we need to resort to the concept of subgradient. We hence have

$$s_i = \boldsymbol{H}_{i:}\boldsymbol{x} + g_i = \begin{cases} \lambda & x_i < 0 \\ -\lambda & x_i > 0 \\ \in [-\lambda, \lambda] & x_i = 0 \end{cases}, \tag{2.50}$$

where $\boldsymbol{H}_{i:}$ is the i-th row of $\boldsymbol{H}$, similarly $\boldsymbol{H}_{:i}$ is the i-th column of $\boldsymbol{H}$. The algorithm proceeds as follows. We iteratively select a variable x_1 which violates the optimality condition in Equation (2.50). In an iteration, x_1 is updated analytically as in Equation (2.46). If all variables satisfy the optimality condition, the algorithm terminates. In order to maximize the effort of violating variable selection, the one which violates the optimality condition the most should be preferred. The extent of violation is measured by the difference, as given below, between s_i and its desired values:

$$d_i = \begin{cases} |s_i - \lambda| & x_i < 0 \\ |s_i + \lambda| & x_i > 0 \\ |s_i| - \lambda & x_i = 0 \end{cases}. \tag{2.51}$$

Update $\boldsymbol{s}$ and Compute b_1

In each iteration after updating x_1, the vector $\boldsymbol{s}$ needs to be updated in order to obtain the optimality condition and select the new violating variable x_1. Intuitively, $\boldsymbol{s}$ can be updated by its definition in Equation (2.50). However, it would take linear time to update each element s_i. In fact, if we keep a record of its previous value (denoted by s'_i), s_i can be updated in constant time. We denote by $\boldsymbol{x}'$ the coefficients before updating x_1, and by $\boldsymbol{x}$ the coefficients after updating x_1. We know that $s'_i = h_{i1}x'_1 + \boldsymbol{H}_{i\mathcal{P}}\boldsymbol{x}'_{\mathcal{P}} + g_i$, $s_i = h_{i1}x_1 + \boldsymbol{H}_{i\mathcal{P}}\boldsymbol{x}_{\mathcal{P}} + g_i$, and

$\boldsymbol{H}_{i\mathcal{P}}\boldsymbol{x}'_{\mathcal{P}} = \boldsymbol{H}_{i\mathcal{P}}\boldsymbol{x}_{\mathcal{P}}$. We thus can update s_i by the following equation which takes constant time:

$$s_i = h_{i1}(x_1 - x'_1) + s'_i. \tag{2.52}$$

Similar idea also applies to the computation of b_1 before updating x_1. According to the definition in Equation (2.40), b_1 can be updated in linear time. However, it can actually be updated in constant time as well. We know that $s'_1 = h_{11}x'_1 + \boldsymbol{H}_{1\mathcal{P}}\boldsymbol{x}'_{\mathcal{P}} + g_1 = h_{11}x' + b_1$. We thus can update b_1 in constant time:

$$b_1 = s'_1 - h_{11}x'. \tag{2.53}$$

Initialize $\boldsymbol{x}$ and $\boldsymbol{s}$

Before the iterative update of the method, $\boldsymbol{x}$ and $\boldsymbol{s}$ need to be initialized. We can initialize $\boldsymbol{x}$ and $\boldsymbol{s}$ by zeros and $\boldsymbol{g}$, respectively. This initialization makes the following iterative update very efficient. This is because $\boldsymbol{x}$ is eventually sparse, which means that $\boldsymbol{x} = \boldsymbol{0}$ is a very good approximate. This initialization strategy may also apply to many other sparse coding methods, for example active-set methods.

4.9 Decomposition Method for NNQP

Now we concisely derive the decomposition method for NNQP (Equation (3.9)). The decomposed objective of NNQP can be formulated into

$$f(\boldsymbol{x}_{\mathcal{A}}) = \frac{1}{2}\boldsymbol{x}_{\mathcal{A}}^{\mathrm{T}}\boldsymbol{H}_{\mathcal{A}\mathcal{A}}\boldsymbol{x}_{\mathcal{A}} + (\boldsymbol{H}_{\mathcal{A}\mathcal{P}}\boldsymbol{x}_{\mathcal{P}} + \boldsymbol{g}_{\mathcal{A}})^{\mathrm{T}}\boldsymbol{x}_{\mathcal{A}} + constant. \tag{2.54}$$

Since this objective is constrained only by nonnegativity, its SMO case also has only *one* variable in $\mathcal{A}$. Without loss of generality, we denote such variable as x_1 as above. This results in the following problem:

$$\min_{x_1} f(x_1) = \frac{1}{2}h_{11}x_1^2 + b_1x_1 \text{ s.t. } x_1 \geq 0, \tag{2.55}$$

where $b_1 = \boldsymbol{H}_{1\mathcal{P}}\boldsymbol{x}_{\mathcal{P}} + g_1$. It is easy to obtain the analytical solution to Equation (2.55):

$$x_1^* = \begin{cases} \frac{-b_1}{h_{11}} & \text{if } b_1 \leq 0 \\ 0 & \text{otherwise} \end{cases}. \tag{2.56}$$

The Lagrangian function of Equation (3.9) is

$$L(\boldsymbol{x}, \boldsymbol{s}) = \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x} + \boldsymbol{g}^{\mathrm{T}}\boldsymbol{x} - \boldsymbol{s}^{\mathrm{T}}\boldsymbol{x}, \tag{2.57}$$

where $\boldsymbol{s}$ is the vector of dual variables. Therefore, the corresponding KKT conditions of Equation (3.9) are

$$\begin{cases} \boldsymbol{s} = \boldsymbol{H}\boldsymbol{x} + \boldsymbol{g} \\ \boldsymbol{s} * \boldsymbol{x} = 0 \\ \boldsymbol{x} \geq 0 \\ \boldsymbol{s} \geq 0. \end{cases} \tag{2.58}$$

From the KKT conditions, we can find that $s_i = \boldsymbol{H}_{i:}\boldsymbol{x} + g_i$ is the Lagrangian multiplier of the i-th primal variable, x_i. For the optimal x_i, the corresponding s_i must fulfil the following conditions:

$$\begin{cases} s_i * x_i = 0 \\ s_i \geq 0 \end{cases} . \tag{2.59}$$

The SMO-based NNQP works in an iterative procedure. Before the iterative loop, $\boldsymbol{x}$ can be initialized by zeros, and $\boldsymbol{s}$ is therefore initialized with $\boldsymbol{g}$. After that, variables violating the KKT conditions in Equation (2.59) are updated in an iterative loop until all variables fulfil the KKT conditions. In each iteration, we need to find one variable which violates the KKT condition in order to update by Equation (2.56); before updating x_1, b_1 can be computed using Equation (2.53); after updating x_1, $\boldsymbol{s}$ can be updated as in Equation (2.52). In our implementation, we select the variable which violates the KKT conditions the most. The magnitude of violation is measured by the following equation:

$$d_i = \begin{cases} 0 & \text{if } x_i = 0 \text{ and } s_i \geq 0 \\ |s_i| & \text{if } x_i = 0 \text{ and } s_i < 0 \\ |s_i| & \text{if } x_i > 0 \end{cases} . \tag{2.60}$$

4.10 Kernel Extensions

Two main advantages of a kernel method are given as follows. First, it linearize complex patterns in a higher-dimensional feature space. Second, it is dimension-free in optimization and decision making. Dimension-free means that the computation is not explicitly affected by the number of features. The methods require only the inner products between multivariate samples (that is $\boldsymbol{d}_i^{\mathrm{T}}\boldsymbol{d}_j$ where $\boldsymbol{d}_i, \boldsymbol{d}_j \in \mathbb{R}^m$). The number of features, m, does not affect the time complexity and spacial complexity at all. In the following, we show that the sparse coding techniques can be extended to kernel versions.

As the optimizations of l_1QP and NNQP only require inner products between the instances instead of the original data, our active-set algorithms can be naturally extended to solve the kernel sparse coding problem by replacing inner products with kernel matrices. The NS decision rule used in Algorithm 1 also requires only inner products. And the weighted K-NN rule only needs the sparse coefficient vector and class information. Therefore, the classification approach in Algorithm 1 can be extended to kernel version. For narrative convenience, we also denote the classification approaches using l_1LS, NNLS, and l_1NNLS sparse coding as

l_1LS, NNLS, and l_1NNLS, respectively. Prefix "K" is used for kernel versions.

5 The Performance of Sparse Coding

5.1 The Performance of Active-Set Sparse Coding for Classification

Two high-throughput biological data, including a microarray gene expression data set and a protein mass spectrometry data set, are used to test the performance of our sparse coding based classification approach. The microarray data set is a collection of gene expression profiles of breast cancer subtypes [29]. This data set includes 158 tumor samples from five subtypes measured on 13582 genes. The mass spectrometry data set is composed of 332 samples from normal class and prostate cancer class [30]. Each sample has 15154 features, that is the mass-to-charge ratios.

When dictionary learning was not involved, the dictionary was "lazily" composed by all the training available instances. In our experiment, the active-set optimization methods for l_1LS, NNLS, and l_1NNLS were tested. The weighted K-NN rule and NS rule, mentioned in Algorithm 1, were compared. We set K in the K-NN rule to the number of all training instances, which is an extreme case as opposite to the NN rule. Linear and *radial basis function* (RBF) kernels were employed. We compared our active-set algorithms with the interior-point [21] method and proximal [23] method for l_1LS sparse coding (abbreviated by l_1LS-IP and l_1LS-PX). Benchmark classifiers, including k-NN and SVM using RBF kernel, were compared. We employed four-fold *cross-validation* to partition a data set into training sets and test sets. All the classifiers ran on the same training and test splits for fair comparison. We performed 20 runs of cross-validation and recorded the averages and standard deviations. Line or grid search was used to select the parameters of a classifiers.

We use accuracy to measure the classification performance. The accuracy is defined as the ratio of the number of correctly predicted test samples to the number of all test samples. The average accuracies of all classifiers with the corresponding standard deviations on both data sets are compared in Figure 1, from which we have four observations. First, the weighted K-NN rule obtained comparable accuracies with the NS rule. The advantage of the K-NN rule over the NS rule is that the former predicts the class labels based on the sparse coefficient vector solely, while the latter has to use the training data to compute regression residuals. Therefore, the K-NN rule is more efficient and should be preferred. Second, on the Prostate data, the sparse coding method l_1LS and Kl_1LS achieved the best accuracy. This convinces us that sparse coding based classifiers can be very effective for classifying high-throughput biological data. Third, the non-negative models including NNLS, l_1NNLS and their kernel extensions achieved competitive accuracies with the state-of-the-art SVM on both data set. Fourth, the l_1LS sparse coding using our active-set algorithm had the same accuracy as that using the interior-point algorithm and proximal algorithm on Breast data. But on Prostate data, the proximal method yielded a worse accuracy. This implies that our active-set method converges to the global minima as the interior-point method, while performance may be deteriorated by the approximate solution obtained by the proximal method in practice.

The mean running time (in seconds) of cross-validation are shown in Figure 2. For better comparison,

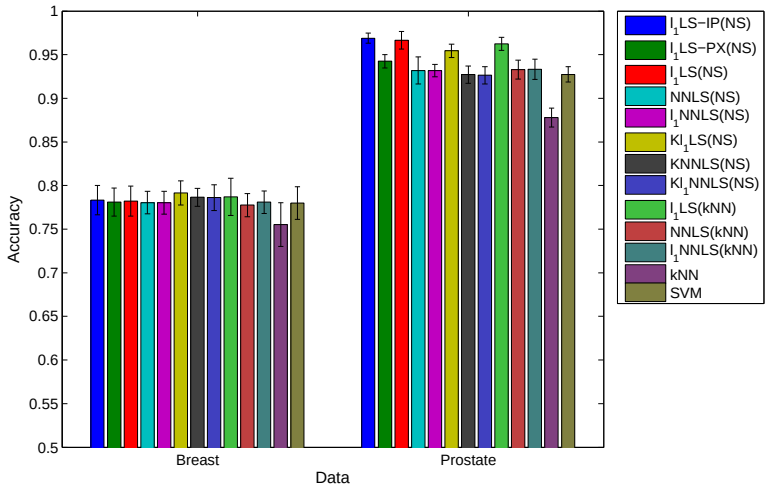


Figure 2.1: Accuracies of sparse coding and benchmark approaches. This a color figure, thus may affect the readability if printed in grayscale. The order of the bars, from left to right, in the figure, is the same as these from top to bottom in the legend. In the legend, the K-NN rule and NS rule is indicated in the corresponding parentheses for sparse coding classifiers. IP and PX are the abbreviations of the interior-point and proximal methods, respectively. The rest sparse coding models use active-set method without explicit notation.

logarithm of base two was taken on the results. First of all, we can clearly see that the interior-point method is very slow for the l_1LS sparse coding. Second, our active-set method is more efficient than the proximal method on Breast data. This is because i) active-set methods are usually the fastest ones for quadratic and linear programmes of small and medium sizes; and ii) expensive computations, like solving systems of linear equations, can be shared in the active-set method. Third, NNLS and l_1NNLS have the same time-complexity. This is reasonable, because both can be formulated as NNQP problems. These non-negative models are much simpler and faster than the non-smooth l_1LS model. Hence, if similar performance can be obtained by l_1LS and the non-negative models in an application, we should give preference to NNLS and l_1NNLS.

5.2 The Performance of Decomposition Method

We tested our SMO methods on the large-scale microarray data given in [31]. This data set has 5456 samples including healthy tissue samples and cancer samples from 13 different tissue types. The number of dimensions (genes) of each sample is 9471. We randomly selected 100 samples as test set, and used the remaining 5356 samples as training set. In few cases where the computation was very costly, we only used 10 test samples. We put all the training samples in the dictionary $\boldsymbol{A}_{9471\times5356}$. Therefore, the Hessian is of size 5356 by 5356. We used linear kernel in our experiments. We compared our SMO methods with proximal method and active-set methods. Due to the large numbers of samples and features, the interior-point method proposed in [21] crashed our computer (the implementation in [21] is not dimension-free). We recorded the wall-clock computing time, in seconds, and the number of iterations of the tests, as the value of λ increased. The average results of each method are given in Tables 2.1 and 2.2. The corresponding sparsity is also given in the first

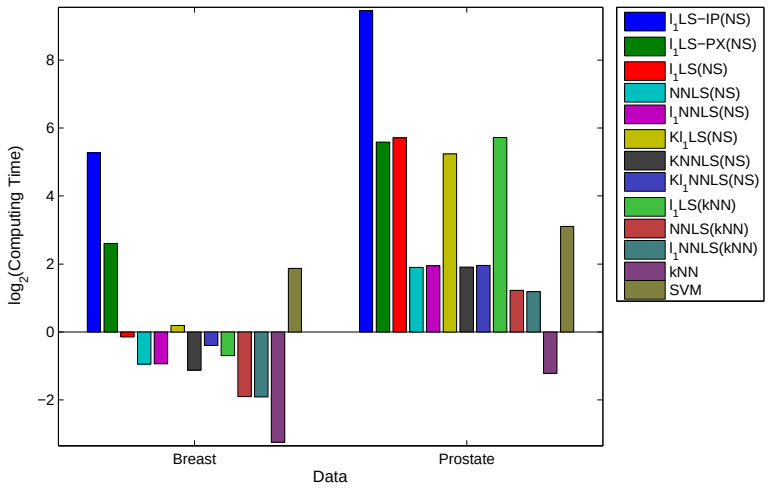


Figure 2.2: Computing time of sparse coding and benchmark methods. This a color figure, thus may affect the readability if printed in grayscale. The order of the bars, from left to right, in the figure, is the same as these from top to bottom in the legend. In the legend, the K-NN rule and NS rule is indicated in the corresponding parentheses for sparse coding classifiers. IP and PX are the abbreviations of the interior-point and proximal methods, respectively. The rest sparse coding models use active-set method without explicit notation.

table. The sparsity is the percentage of zeros in a coefficient vector. It is defined as $\frac{\text{sum}(\boldsymbol{x}<\varepsilon \max(\boldsymbol{x}))}{k}$, where $\boldsymbol{x}$ is the sparse coefficient vector, k is the length of $\boldsymbol{x}$, and we set $\varepsilon = 0.001$ in our experiment.

From Table 2.1, we have the following observations. First, for all methods the computing time and numbers of iterations decrease gradually as the value of λ increases. Second, for the non-smooth l_1QP model, our SMO method is much faster than the proximal and active-set methods. Third, for the smooth NNQP model, SMO is also efficient, though the active-set method keeps its efficiency. Fourth, the sparsity increases as the value of λ rises. When $\lambda = 0$, only the non-negativity induces the sparsity in the NNQP model. We can see that very high sparsity can be obtained by using the non-negativity solely.

From Table 2.2, the active-set algorithms converge to the optimal solution in a few iterations, but the computational cost of each iteration is expensive, since the Hessian matrix is involved in the computation. Our method and the proximal method have a larger number of iterations, and the computational cost of each iteration is very low. Both methods have closed-form update solution. The operation on the Hessian is avoided in our SMO methods. Finally, we should mention that our SMO methods can obtain identical solutions to the active-set methods, which corroborates that the decomposition methods converge well. However, the proximal method can only obtain approximate results in many cases, because it is a first-order method that needs a large number of iterations to converge.

Additionally, we tested the prediction accuracies of our l_1QP and NNQP models for classifying the microarray gene profiles. We used our SMO methods in the optimizations. The mean accuracies of 10-fold cross-validation are given in Table 2.3. We can see that both models obtained very high accuracies. The NNQP model with $\lambda = 0$ obtained the best result on this data. This observation suggests that we need to

Table 2.1: Mean computing time (in seconds) of each sample when using 5356 samples as training set.

λ	l_1QP				NNQP		
	SMO	proximal	active-set	sparsity	SMO	active-set	sparsity
0	-	-	-	-	17.99	0.49	0.9954
0.0001	273.78	554.37	-	0.9478	17.94	0.48	0.9954
0.001	22.30	557.20	-	0.9950	18.02	0.49	0.9954
0.01	20.75	558.47	-	0.9956	17.66	0.49	0.9956
0.1	16.20	550.69	-	0.9968	13.67	0.34	0.9968
0.2	12.93	635.15	-	0.9975	10.99	0.29	0.9975
0.3	10.02	615.11	1669.63	0.9980	8.48	0.22	0.9980
0.4	7.83	572.55	1426.41	0.9984	6.58	0.17	0.9984
0.5	6.39	513.75	1157.20	0.9987	5.42	0.14	0.9987
0.6	5.17	441.12	923.95	0.9989	4.40	0.12	0.9989
0.7	3.91	357.81	684.89	0.9992	3.31	0.10	0.9992
0.8	2.86	254.41	553.94	0.9994	2.42	0.07	0.9994
0.9	1.70	97.00	395.52	0.9996	1.44	0.05	0.9996

Table 2.2: Mean number of iterations of each sample when using 5356 samples as training set.

λ	l_1QP			NNQP	
	SMO	proximal	active-set	SMO	active-set
0	-	-	-	53033	29.37
0.0001	682153	29250	-	52994	29.36
0.001	55612	29427	-	52981	29.30
0.01	51438	29451	-	51439	28.23
0.1	40068	29302	-	40068	20.07
0.2	31962	34334	-	31962	15.43
0.3	24899	33212	13.30	24899	12.20
0.4	19348	30902	11.40	19348	9.80
0.5	15927	27752	9.40	15927	8.22
0.6	12853	23834	7.80	12853	6.88
0.7	9680	19236	6.00	9680	5.51
0.8	7092	13144	5.00	7092	4.30
0.9	4210	5152	3.80	4210	3.09

try the easier NNQP model first before resorting to l_1QP model, when applying sparse coding techniques in other problems, particularly in classification.

6 Bayesian Dictionary Learning

We pursue our dictionary-learning-based approach for biological data, based on the following two motivations. First, the sparse-coding-only approach, which places all training samples in the dictionary, is a lazy

Table 2.3: Mean prediction accuracies of the l_1QP and NNQP models using 10-fold cross-validation.

λ	l_1QP	NNQP
0	-	0.9762
0.5	0.9727	0.9727
0.9	0.9654	0.9654

learning, thus the optimization can be slow for large training set. Therefore, learning a concise dictionary is more efficient for future real-time applications. Second, dictionary learning can capture hidden key factors which correspond to biological pathways, and the classification performance may hence be improved. In the following, we first give the dictionary learning models using Gaussian prior and uniform prior, respectively. Next, we give the classification method based on dictionary learning. We then address the generic optimization framework of dictionary learning. Finally, we show that the kernel versions of our dictionary learning models and classification approach can be easily obtained.

6.1 Dictionary Learning Models

Now we give our dictionary learning models using Gaussian prior and uniform prior over the dictionary atoms, respectively. Both priors aims to get rid off the arbitrary scale interchange between dictionary and coefficient. Suppose $\boldsymbol{D}_{m\times n}$ is the data of n training instances, and the dictionary $\boldsymbol{A}$ to be learned has k atoms. If the Gaussian prior in Equation (2.6) is used on the dictionary atom, our dictionary learning models of l_1LS, NNLS, and l_1NNLS are expressed as follow, respectively:

$$l_1LS: \min_{\boldsymbol{A},\boldsymbol{Y}} \frac{1}{2}\|\boldsymbol{D}-\boldsymbol{A}\boldsymbol{Y}\|_F^2 + \frac{\alpha}{2}\text{trace}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}) + \lambda\sum_{i=1}^{n}\|\boldsymbol{y}_i\|_1, \tag{2.61}$$

$$\begin{aligned} NNLS: \min_{\boldsymbol{A},\boldsymbol{Y}} & \frac{1}{2}\|\boldsymbol{D}-\boldsymbol{A}\boldsymbol{Y}\|_F^2 + \frac{\alpha}{2}\text{trace}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}) \\ \text{s.t. } & \boldsymbol{Y} \geq 0, \end{aligned} \tag{2.62}$$

and

$$\begin{aligned} l_1NNLS: \min_{\boldsymbol{A},\boldsymbol{Y}} & \frac{1}{2}\|\boldsymbol{D}-\boldsymbol{A}\boldsymbol{Y}\|_F^2 + \frac{\alpha}{2}\text{trace}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}) + \sum_{i=1}^{n}\boldsymbol{\lambda}^{\mathrm{T}}\boldsymbol{y}_i \\ \text{s.t. } & \boldsymbol{Y} \geq 0. \end{aligned} \tag{2.63}$$

The strength of the Gaussian prior based dictionary learning is that, it is flexible to control the scales of dictionary atoms by tuning α. However, l_1LS and l_1NNLS have two model parameters (that is α and λ), which increase the model selection burden in practice.

Alternatively, in order to eliminate the parameter α, we design an uniform prior over the dictionary which is expressed as

$$p(\boldsymbol{a}_i) = \begin{cases} \theta & \text{if } \|\boldsymbol{a_i}\|_2 = 1, \\ 0 & \text{otherwise,} \end{cases} \tag{2.64}$$

where $\theta \in (0,1)$ is a constant. That is, the feasible region of the dictionary atoms is a hypersphere centered at origin with unit radius, and all the feasible atoms have equal probability. The corresponding dictionary learning models are given in the following equations, respectively:

$$\begin{aligned} l_1LS: \min_{\boldsymbol{A},\boldsymbol{Y}} \frac{1}{2}\|\boldsymbol{D}-\boldsymbol{A}\boldsymbol{Y}\|_F^2 + \lambda\sum_{i=1}^{n}\|\boldsymbol{y}_i\|_1 \\ \text{s.t. } \boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{a}_i = 1, \quad i = 1,\cdots,k, \end{aligned} \tag{2.65}$$

$$\begin{aligned} NNLS: \min_{\boldsymbol{A},\boldsymbol{Y}} \frac{1}{2}\|\boldsymbol{D}-\boldsymbol{A}\boldsymbol{Y}\|_F^2 \\ \text{s.t. } \boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{a}_i = 1, \quad i = 1,\cdots,k; \quad \boldsymbol{Y} \geq 0, \end{aligned} \tag{2.66}$$

and

$$\begin{aligned} l_1NNLS: \min_{\boldsymbol{A},\boldsymbol{Y}} \frac{1}{2}\|\boldsymbol{D}-\boldsymbol{A}\boldsymbol{Y}\|_F^2 + \sum_{i=1}^{n}\boldsymbol{\lambda}^{\mathrm{T}}\boldsymbol{y}_i \\ \text{s.t. } \boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{a}_i = 1, \quad i = 1,\cdots,k; \quad \boldsymbol{Y} \geq 0. \end{aligned} \tag{2.67}$$

6.2 A Generic Optimization Framework for Dictionary Learning

We devise a block-coordinate-descent based algorithms for the optimization of the above six models. The main idea is that $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ and $\boldsymbol{Y}$ are updated alternatingly. In a step, $\boldsymbol{Y}$ is fixed, and the inner product $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$, rather than $\boldsymbol{A}$ itself, is updated; in the next step, $\boldsymbol{Y}$ is updated while fixing $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ (a sparse coding procedure). The above procedure is repeated until the termination conditions are satisfied.

Now, we show that $\boldsymbol{A}$ can be analytically obtained. For normal prior over dictionary atoms, the optimization of finding $\boldsymbol{A}$ in Equations (2.61), (2.62), and (2.63) is to solve

$$\min_{\boldsymbol{A}} f(\boldsymbol{A}) = \frac{1}{2}\|\boldsymbol{D}-\boldsymbol{A}\boldsymbol{Y}\|_F^2 + \frac{\alpha}{2}\mathrm{trace}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}). \tag{2.68}$$

Taking the derivative with respect to $\boldsymbol{A}$ and setting it to zero, we have

$$\frac{\partial f(\boldsymbol{A})}{\partial \boldsymbol{A}} = \boldsymbol{A}\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}} - \boldsymbol{D}\boldsymbol{Y}^{\mathrm{T}} + \alpha\boldsymbol{A} = 0. \tag{2.69}$$

We hence have

$$\boldsymbol{A} = \boldsymbol{D}\boldsymbol{Y}^{\ddagger}, \tag{2.70}$$

where $\boldsymbol{Y}^{\ddagger} = \boldsymbol{Y}^{\mathrm{T}}(\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}} + \alpha\boldsymbol{I})^{-1}$. The inner product $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ can thus be updated by

$$\boldsymbol{R} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} = (\boldsymbol{Y}^{\ddagger})^{\mathrm{T}}\boldsymbol{D}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{Y}^{\ddagger}. \tag{2.71}$$

We also can compute $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{D}$ by

$$\boldsymbol{A}^{\mathrm{T}}\boldsymbol{D} = (\boldsymbol{Y}^{\ddagger})^{\mathrm{T}}\boldsymbol{D}^{\mathrm{T}}\boldsymbol{D}. \tag{2.72}$$

For the uniform prior as in Equation (2.64), updating unnormalized $\boldsymbol{A}$ while fixing $\boldsymbol{Y}$ in Equations (2.65), (2.66), and (2.67) is to solve the generalized least squares:

$$\min_{\boldsymbol{A}} f(\boldsymbol{A}) = \frac{1}{2}\|\boldsymbol{D} - \boldsymbol{A}\boldsymbol{Y}\|_F^2. \tag{2.73}$$

Taking the derivative with respect to $\boldsymbol{A}$ and setting it to zero, we have

$$\boldsymbol{A} = \boldsymbol{D}\boldsymbol{Y}^{\dagger}, \tag{2.74}$$

where $\boldsymbol{Y}^{\dagger} = \boldsymbol{Y}^{\mathrm{T}}(\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}})^{-1}$. The inner products of $\boldsymbol{R} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ and $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{D}$ are computed similarly as for the Gaussian prior. The normalization of $\boldsymbol{R}$ is straightforward. We have $\boldsymbol{R} = \boldsymbol{R}./\sqrt{\mathrm{diag}(\boldsymbol{R})\mathrm{diag}(\boldsymbol{R})^{\mathrm{T}}}$, where $./$ and $\sqrt{\bullet}$ are element-wise operators. Learning the inner product $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ instead of $\boldsymbol{A}$ has the benefits of dimension-free computation and kernelization.

Fixing $\boldsymbol{A}$, $\boldsymbol{Y}$ can be obtained via our active-set method described in Section 4.6. Recall that the sparse coding only requires the inner products $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ and $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{D}$. As shown above, we find that updating $\boldsymbol{Y}$ only needs its previous value and the inner product between training instances.

Due to the above derivation, we have the framework of solving our dictionary learning models as illustrated in Algorithm 4.

6.3 Classification Approach Based on Dictionary Learning

Now, we present the dictionary-learning based classification approach in Algorithm 5. The dictionary learning in the training step should be consistent with the sparse coding in the prediction step. As discussed in the previous section, the sparse coding in the prediction step needs the inner products $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$, $\boldsymbol{B}^{\mathrm{T}}\boldsymbol{B}$ and $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{B}$. Actually, $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{B}$ is either $\boldsymbol{Y}^{\ddagger\mathrm{T}}\boldsymbol{D}^{\mathrm{T}}\boldsymbol{B}$ or $\boldsymbol{Y}^{\dagger\mathrm{T}}\boldsymbol{D}^{\mathrm{T}}\boldsymbol{B}$.

Algorithm 4 Generic Dictionary Learning Framework

Input: $\boldsymbol{K} = \boldsymbol{D}^{\mathrm{T}}\boldsymbol{D}$, dictionary size k, λ
Output: $\boldsymbol{R} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$, $\boldsymbol{Y}$

initialize $\boldsymbol{Y}$ and $\boldsymbol{R} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ randomly;
$r_{prev} = Inf$; {previous residual}

for $i = 1 : maxIter$ **do**
  Update $\boldsymbol{Y}$ by solving the active-set based l_1LS, NNLS, or l_1NNLS sparse coding algorithms;

  if Gaussian prior over $\boldsymbol{A}$ **then**
    Update $\boldsymbol{R} = \boldsymbol{Y}^{\ddagger\mathrm{T}}\boldsymbol{D}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{Y}^{\ddagger}$;
  end if

  if uniform prior over $\boldsymbol{A}$ **then**
    Update $\boldsymbol{R} = \boldsymbol{Y}^{\dagger\mathrm{T}}\boldsymbol{D}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{Y}^{\dagger}$;
    Normalize $\boldsymbol{R}$ by $\boldsymbol{R} = \boldsymbol{R}./\sqrt{\mathrm{diag}(\boldsymbol{R})\mathrm{diag}(\boldsymbol{R})^{\mathrm{T}}}$;
  end if

  if $i == maxIter$ or $i \mod l == 0$ **then**
    {check every l iterations}
    $r_{cur} = f(\boldsymbol{A}, \boldsymbol{Y})$; {current residual of a dictionary learning model}
    if $r_{prev} - r_{cur} \leq \varepsilon$ or $r_{cur} \leq \varepsilon$ **then**
      Break;
    end if
    $r_{prev} = r_{cur}$;
  end if
end for

Algorithm 5 *Dictionary-Learning Based Classification*

Input: $\boldsymbol{D}_{m\times n}$: n training instances, $\boldsymbol{c}$ the class labels, $\boldsymbol{B}_{m\times p}$: p new instances, k: dictionary size
Output: $\boldsymbol{p}$: the predicted class labels of the p new instances

{**Training Step:**}

1. Normalize each training instance to have unit l_2 norm.
2. Learn dictionary inner product $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ and sparse coefficient matrix $\boldsymbol{Y}$ of training instances by Algorithm 4.
3. Train a classifier $f(\theta)$ with parameter θ using $\boldsymbol{Y}$ (in the feature space spanned by columns of $\boldsymbol{A}$).

{**Prediction Step:**}

1. Normalize each new instance to have unit l_2 norm.
2. Obtain the sparse coefficient matrix $\boldsymbol{X}$ of the new instances by solving Equation (2.37), or (2.38).
3. Predict the class labels of $\boldsymbol{X}$ using the classifier $f(\theta)$ learned in the training phase.

6.4 Kernel Extensions

For Gaussian dictionary prior, the l_1LS based kernel dictionary learning and sparse coding are expressed in the following, respectively:

$$\min_{\boldsymbol{A}_\phi, \boldsymbol{Y}} \frac{1}{2}\|\phi(\boldsymbol{D}) - \boldsymbol{A}_\phi \boldsymbol{Y}\|_F^2 + \frac{\alpha}{2}\text{trace}(\boldsymbol{A}_\phi^{\mathrm{T}}\boldsymbol{A}_\phi) + \lambda\|\boldsymbol{Y}\|_1, \tag{2.75}$$

$$\min_{\boldsymbol{X}} \frac{1}{2}\|\phi(\boldsymbol{B}) - \boldsymbol{A}_\phi \boldsymbol{X}\|_F^2 + \lambda\|\boldsymbol{X}\|_1,$$

where $\phi(\bullet)$ is a mapping function. Equations (2.62), (2.63), (2.65), (2.66), (2.67) and their sparse coding models can be kernelized analogously. As we have mentioned already, the optimizations of the six dictionary learning models only involve inner products of instances. Thus, we can easily obtain their kernel extensions by replacing the inner products with kernel matrices. Hereafter, if dictionary learning is employed in sparse representation, then prefix "DL" is used before "l_1LS", "NNLS", and "l_1NNLS". If kernel function other than the linear kernel is used in dictionary learning, then prefix "KDL" is added before them.

6.5 Computational Experiments

Two high-throughput biological data, including a microarray gene expression data set and a protein mass spectrometry data set, are used to test the performance of our dictionary learning methods for dimensionality reduction. The microarray data set is a collection of gene expression profiles of breast cancer subtypes [29]. This data set includes 158 tumor samples from five subtypes measured on 13582 genes. The mass spectrometry data set is composed of 332 samples from normal class and prostate cancer class [30]. Each sample has 15154 features, that is the mass-to-charge ratios. The performance is measured by accuracy and running time in seconds.

The performance of various dictionary learning models with linear and RBF kernels were investigated on both Breast and Prostate data sets. The Gaussian-prior based and uniform-prior based dictionary learning models were also compared. Again, our active-set dictionary learning method was compared with the interior-point [21] and proximal [23] methods. The semi-NMF based on multiplicative update rules [32] is also included in the comparison. As in the experiment of sparse coding in Section 5, four-fold cross-validation was used. All methods ran on the same splits of training and test sets. We performed 20 runs of cross-validation for reliable comparison. After feature extraction by using dictionary learning on the training set, the linear SVM classifier was trained on the reduced training set and was used to predict the class labels of test instances.

In Figure 3, we show the mean accuracy and standard deviation of 20 results for each method. First, we can see that the models with Gaussian prior on dictionary atoms obtained similar accuracies as the uniform prior. Second, with the comparison to sparse coding methods on Breast data as given Figure 1, we can see that dictionary learning increases the prediction accuracy. Third, from the comparison of Figures 3 and 1, we find that the dictionary learning based methods – DL-NNLS and DL-l_1NNLS, obtained similar accuracies as the sparse coding methods – NNLS and l_1NNLS. This convinces us that dictionary learning is a promising

feature extraction technique for high-dimensional biological data. On Prostate data, we can also find that the accuracy obtained by DL-l_1LS is slightly lower than l_1LS. This is may be because the dictionary learning is unsupervised. Fourth, using the model parameters, DL-l_1LS using active-set algorithm obtained higher accuracy than DL-l_1LS-IP and DL-l_1LS-PX on Prostate data. The accuracy of DL-l_1LS is also slightly higher than that of DL-l_1LS-IP on Breast data. Furthermore, the non-negative DL-NNLS yielded the same performance as the well-known semi-NMF, while further corroborating the satisfactory performance of our dictionary learning framework. Finally, the kernel dictionary learning models achieved similar performance as their linear counterparts. We believe that the accuracy could be further improved by a suitably selected kernel.

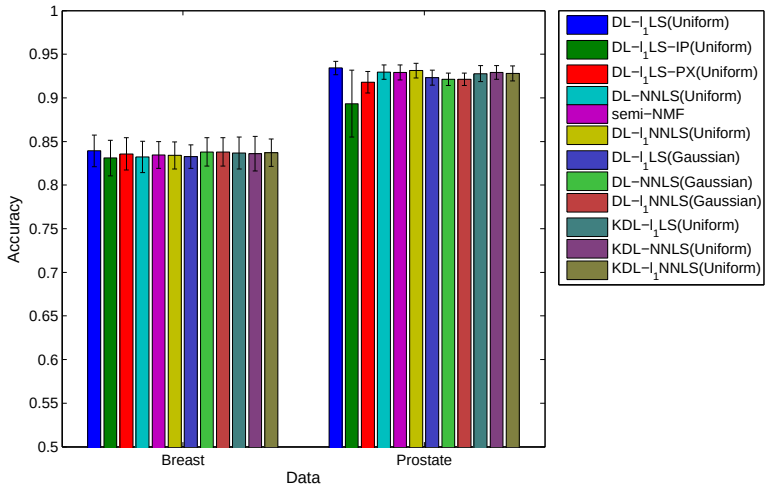


Figure 2.3: Accuracies of dictionary learning approaches. This a color figure, thus may affect the readability if printed in grayscale. The order of the bars, from left to right, in the figure, is the same as these from top to bottom in the legend. In the legend, the Gaussian and uniform priors are indicated in the corresponding parentheses. IP and PX are the abbreviations of the interior-point and proximal methods, respectively. The rest dictionary learning models use active-set method without explicit notation.

We compare the mean computing time of all the feature extraction methods in Figure 4. First, we can see that DL-l_1LS using active-set algorithm is much more efficient than DL-l_1LS-IP, DL-l_1LS-PX, and semi-NMF using multiplicative update rules. Second, the non-negative dictionary learning models are more efficient than the l_1-regularized models. Therefore as in the sparse coding method, priority should be given to the non-negative models when attempting to use dictionary learning in an application.

7 Versatile Sparse Matrix Factorization

We have proposed a generic dictionary learning framework in Section 6.1. Now, we present a more general dictionary learning framework, named *versatile sparse matrix factorization* (VSMF).

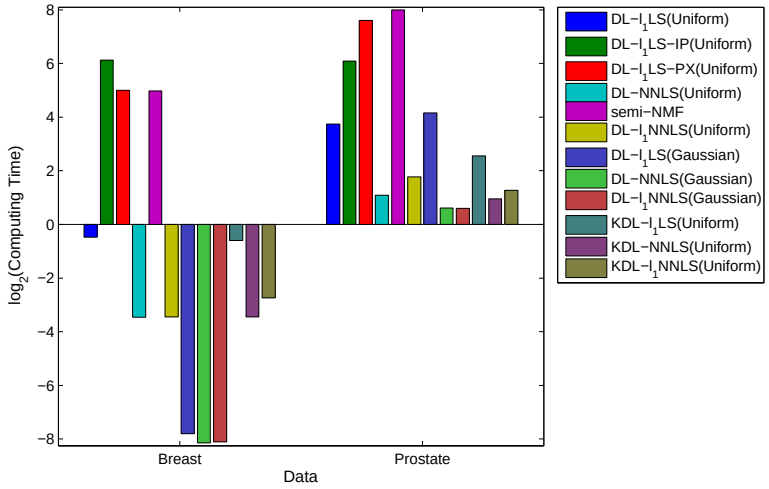


Figure 2.4: Computing time of dictionary learning approaches. This a color figure, thus may affect the readability if printed in grayscale. The order of the bars, from left to right, in the figure, is the same as these from top to bottom in the legend. In the legend, the Gaussian and uniform priors are indicated in the corresponding parentheses. IP and PX are the abbreviations of the interior-point and proximal methods, respectively. The rest dictionary learning models use active-set method without explicit notation.

7.1 Versatile Sparse Matrix Factorization Model

Our VSMF model can be expressed in the following equation:

$$\begin{aligned}\min_{\boldsymbol{A},\boldsymbol{Y}} f(\boldsymbol{A},\boldsymbol{Y}) &= \frac{1}{2}\|\boldsymbol{D}-\boldsymbol{A}\boldsymbol{Y}\|_F^2 + \sum_{i=1}^{k}\left(\frac{\alpha_2}{2}\|\boldsymbol{a}_i\|_2^2 + \alpha_1\|\boldsymbol{a}_i\|_1\right) \\ &\quad + \sum_{i=1}^{n}\left(\frac{\lambda_2}{2}\|\boldsymbol{y}_i\|_2^2 + \lambda_1\|\boldsymbol{y}_i\|_1\right) \\ \text{s.t.} &\begin{cases} \text{if } t_1 = 1 \quad \boldsymbol{A} \geq 0 \\ \text{if } t_2 = 1 \quad \boldsymbol{Y} \geq 0 \end{cases},\end{aligned} \tag{2.76}$$

where, parameter $\alpha_1 \geq 0$ controls the sparsity of the basis vectors; parameter $\alpha_2 \geq 0$ controls the smoothness and scale of the basis vectors; parameter $\lambda_1 \geq 0$ controls the sparsity of the coefficient vectors; parameter $\lambda_2 \geq 0$ controls the smoothness of the coefficient vectors; and parameters t_1 and t_2 are boolean variables (0: false, 1: true) that indicate if non-negativity should be enforced on $\boldsymbol{A}$ and $\boldsymbol{Y}$, respectively.

One advantage of VSMF is that both l_1 and l_2-norms can be used on both basis matrix and coefficient matrix. In VSMF, l_1-norms are used to induce sparse basis vectors and coefficient vectors. However, the drawback of l_1-norm is that correlated variables may not be simultaneously non-zero in the induced sparse result. This is because l_1-norm is able to produce sparse but non-smooth result. It is known that l_2-norm is able to obtain smooth but not sparse result. It has been demonstrated that correlated variables can be selected or removed simultaneously via combining both norms [33]. In addition to the smoothness of l_2-norm, another

benefit of l_2-norm is that the scale of each vector can be restricted. This can avoid the scale interchange between the basis matrix and the coefficient matrix. Another advantage of VSMF is that the non-negativity constraint can be switched off/on for either basis matrix or coefficient matrix. If the training data are non-negative, it is usually necessary that the basis matrix should be non-negative as well. In some situations, non-negativity is also needed on the coefficient matrix for better performance and better interpretability of results.

For the convenience of discussion, we summarize the existing sparse matrix factorization models in Table 2.4. It is impossible to enumerate all existing works in this direction, therefore all models mentioned in this table are the most representative ones. The training data $\boldsymbol{D}$ must be non-negative for the standard NMF and sparse NMF. For sparse NMF, α and λ are two non-negative parameters. For kernel NMF and DL-l_1LS, $\phi(\cdot)$ is a function that maps the training samples into a high-dimensional feature space. $\phi(\boldsymbol{D})$ is the training samples in this feature space. $\boldsymbol{A}_\phi$ is the basis matrix in this feature space. It can be easily seen that the standard NMF, semi-NMF, and sparse-NMF are special cases of VSMF. If $\alpha_1 = \alpha_2 = \lambda_1 = \lambda_2 = 0$ and $t_1 = t_2 = 1$, VSMF is reduced to the standard NMF proposed in [13]. If $\alpha_1 = \alpha_2 = \lambda_1 = \lambda_2 = 0$ and $t_1 = 0$ and $t_2 = 1$, then VSMF becomes semi-NMF proposed in [32]. If $\alpha_1 = \lambda_2 = 0$, $\alpha_2, \lambda_1 \neq 0$, and $t_1 = t_2 = 1$, then VSMF is equivalent to the sparse-NMF proposed in [34]. When α_1 is set to zero, VSMF can be kernelized (see Section 6.1 and [35]).

Table 2.4: The existing NMF and SR models.

NMF/SR	Equation
Standard NMF [13]	$\min_{\boldsymbol{A},\boldsymbol{Y}} \frac{1}{2}\|\boldsymbol{D}-\boldsymbol{A}\boldsymbol{Y}\|_F^2$ s.t. $\boldsymbol{A},\boldsymbol{Y} \geq 0$
Semi-NMF [32]	$\min_{\boldsymbol{A},\boldsymbol{Y}} \frac{1}{2}\|\boldsymbol{D}-\boldsymbol{A}\boldsymbol{Y}\|_F^2$ s.t. $\boldsymbol{Y} \geq 0$
Sparse NMF [34]	$\min_{\boldsymbol{A},\boldsymbol{Y}} \frac{1}{2}\|\boldsymbol{D}-\boldsymbol{A}\boldsymbol{Y}\|_F^2 + \frac{\alpha}{2}\sum_{i=1}^k \|\boldsymbol{a}_i\|_2^2 + \frac{\lambda}{2}\sum_{i=1}^n \|\boldsymbol{y}_i\|_1^2$ s.t. $\boldsymbol{A},\boldsymbol{Y} \geq 0$
Kernel NMF [36, 35]	$\min_{\boldsymbol{A}_\phi,\boldsymbol{Y}} \frac{1}{2}\|\phi(\boldsymbol{D})-\boldsymbol{A}_\phi\boldsymbol{Y}\|_F^2 + \frac{\alpha}{2}\sum_{i=1}^k \|\phi(\boldsymbol{a}_i)\|_2^2 + \frac{\lambda}{2}\sum_{i=1}^n \|\boldsymbol{y}_i\|_1$ s.t. $\boldsymbol{Y} \geq 0$
l_1-SR [35]	$\min_{\boldsymbol{A}_\phi,\boldsymbol{Y}} \frac{1}{2}\|\phi(\boldsymbol{D})-\boldsymbol{A}_\phi\boldsymbol{Y}\|_F^2 + \frac{\alpha}{2}\sum_{i=1}^k \|\phi(\boldsymbol{a}_i)\|_2^2 + \frac{\lambda}{2}\sum_{i=1}^n \|\boldsymbol{y}_i\|_1$

Sparse matrix factorization is a low-rank approximation problem. The number of ranks, that is k, is crucial for a good performance of an analysis. Selecting k is still an open problem in both statistical inference and machine learning. We propose an adaptive rank selection method for VSMF. We base our idea on the sparsity of columns of $\boldsymbol{A}$ and $\boldsymbol{Y}$. We first set k to a relatively large integer. During the optimization of VSMF, if a column of $\boldsymbol{A}$ or a row of $\boldsymbol{Y}$ is null due to the sparsity controlled by the corresponding parameters, then both of the column of $\boldsymbol{A}$ and the row of $\boldsymbol{Y}$ corresponding to this null factor are removed. Therefore, k is reduced. When the optimization terminates, we can obtain the correct k corresponding to the current sparsity controlling parameters.

7.2 Optimization

Like most of NMF and SR models, the optimization of VSMF is non-convex (an optimization is said to be convex if and only if both of the objective and constraints are convex). The most popular scheme to optimize these models are the block-coordinate descent method [27]. The basic idea of this scheme is in the following.

$\boldsymbol{A}$ and $\boldsymbol{Y}$ are updated iteratively and alternatingly. In each iteration, $\boldsymbol{A}$ is updated while keeping $\boldsymbol{Y}$ fixed; then $\boldsymbol{A}$ is fixed and $\boldsymbol{Y}$ is updated. Based on this scheme, we devise the multiplicative update rules and active-set algorithms for VSMF. These two algorithms are given below.

Multiplicative Update Rules

If both $\boldsymbol{A}$ and $\boldsymbol{Y}$ are non-negative, we can equivalently rewrite $f(\boldsymbol{A},\boldsymbol{Y})$ in Equation (3.11) as

$$\frac{1}{2}\|\boldsymbol{D}-\boldsymbol{A}\boldsymbol{Y}\|_F^2+\frac{\alpha_2}{2}\text{trace}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A})+\alpha_1\text{trace}(\boldsymbol{E}_1^{\mathrm{T}}\boldsymbol{A})+\frac{\lambda_2}{2}\text{trace}(\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{Y})+\lambda_1\text{trace}(\boldsymbol{E}_2^{\mathrm{T}}\boldsymbol{Y}), \tag{2.77}$$

where $\boldsymbol{E}_1\in\{1\}^{m\times k}$, and $\boldsymbol{E}_2\in\{1\}^{k\times n}$. Fixing $\boldsymbol{A}$, and updating $\boldsymbol{Y}$ can hence be expressed as

$$\begin{aligned}&\min_{\boldsymbol{Y}} f(\boldsymbol{Y})=\frac{1}{2}\|\boldsymbol{D}-\boldsymbol{A}\boldsymbol{Y}\|_F^2+\frac{\lambda_2}{2}\text{trace}(\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{Y})+\lambda_1\text{trace}(\boldsymbol{E}_2^{\mathrm{T}}\boldsymbol{Y})\\&\text{s.t. } \boldsymbol{Y}\geq 0.\end{aligned} \tag{2.78}$$

Similarly, fixing $\boldsymbol{Y}$, and updating $\boldsymbol{A}$ can be expressed as

$$\begin{aligned}&\min_{\boldsymbol{A}} f(\boldsymbol{A})=\frac{1}{2}\|\boldsymbol{D}-\boldsymbol{A}\boldsymbol{Y}\|_F^2+\frac{\alpha_2}{2}\text{trace}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A})+\alpha_1\text{trace}(\boldsymbol{E}_1^{\mathrm{T}}\boldsymbol{A})\\&\text{s.t. } \boldsymbol{A}\geq 0.\end{aligned} \tag{2.79}$$

We design the following multiplicative update rules for VSMF model in the case of $t_1=t_2=1$:

$$\begin{cases}\boldsymbol{A}=\boldsymbol{A}*\frac{\boldsymbol{D}\boldsymbol{Y}^{\mathrm{T}}}{\boldsymbol{A}\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}}+\alpha_2\boldsymbol{A}+\alpha_1}\\ \boldsymbol{Y}=\boldsymbol{Y}*\frac{\boldsymbol{A}^{\mathrm{T}}\boldsymbol{D}}{\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{Y}+\lambda_2\boldsymbol{Y}+\lambda_1}\end{cases}, \tag{2.80}$$

where $\boldsymbol{A}*\boldsymbol{B}$ and $\frac{\boldsymbol{A}}{\boldsymbol{B}}$ are element-wise multiplication and division between matrix $\boldsymbol{A}$ and $\boldsymbol{B}$, respectively. This algorithm is a gradient-descent based method in essence. Both rules are derived in the following.

For Equation (2.78), the first-order update rule of $\boldsymbol{Y}$ should be generally

$$\boldsymbol{Y}=\boldsymbol{Y}-\boldsymbol{\eta}_2*\frac{\partial f(\boldsymbol{Y})}{\partial \boldsymbol{Y}}. \tag{2.81}$$

where matrix $\boldsymbol{\eta}_2$ is the step size of the update rule. We take the derivative of $f(\boldsymbol{Y})$, in Equation (2.78), with respect to $\boldsymbol{Y}$:

$$\frac{\partial f(\boldsymbol{Y})}{\partial \boldsymbol{Y}}=\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{Y}-\boldsymbol{A}^{\mathrm{T}}\boldsymbol{D}+\lambda_2\boldsymbol{Y}+\lambda_1\boldsymbol{E}_2. \tag{2.82}$$

And we let the step size $\boldsymbol{\eta}_2$ to be

$$\boldsymbol{\eta}_2 = \frac{\boldsymbol{Y}}{\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{Y} + \lambda_2\boldsymbol{Y} + \lambda_1\boldsymbol{E}_2}. \tag{2.83}$$

Substituting Equations (2.82) and (2.83) into Equation (2.81), we have

$$\boldsymbol{Y} = \boldsymbol{Y} * \frac{\boldsymbol{A}^{\mathrm{T}}\boldsymbol{D}}{\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{Y} + \lambda_2\boldsymbol{Y} + \lambda_1\boldsymbol{E}_2}. \tag{2.84}$$

Similarly, for Equation (2.79), the first-order update rule of $\boldsymbol{A}$ should be generally

$$\boldsymbol{A} = \boldsymbol{A} - \boldsymbol{\eta}_1 * \frac{\partial f(\boldsymbol{A})}{\partial \boldsymbol{A}}. \tag{2.85}$$

We take the derivative of $f(\boldsymbol{A})$, in Equation (2.79), with respect to $\boldsymbol{A}$:

$$\frac{\partial f(\boldsymbol{A})}{\partial \boldsymbol{A}} = \boldsymbol{A}\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}} - \boldsymbol{D}^{\mathrm{T}}\boldsymbol{Y} + \alpha_2\boldsymbol{A} + \alpha_1\boldsymbol{E}_1. \tag{2.86}$$

And we let the step size to be

$$\boldsymbol{\eta}_1 = \frac{\boldsymbol{A}}{\boldsymbol{A}\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}} + \alpha_2\boldsymbol{A} + \alpha_1\boldsymbol{E}_1}. \tag{2.87}$$

Substituting Equations (2.86) and (2.87) into Equation (2.85), we have

$$\boldsymbol{A} = \boldsymbol{A} * \frac{\boldsymbol{D}\boldsymbol{Y}^{\mathrm{T}}}{\boldsymbol{A}\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}} + \alpha_2\boldsymbol{A} + \alpha_1}. \tag{2.88}$$

If we let $\alpha_1 = \alpha_2 = \lambda_1 = \lambda_2 = 0$, then the update rules in Equations (2.80) becomes the update rules of the standard NMF [37]. We can find that enforcing sparsity and smoothness on both basis matrix and coefficient matrix does not increase the time-complexity.

Active-Set Quadratic Programming

The multiplicative update rules above only work under the condition that both $\boldsymbol{A}$ and $\boldsymbol{Y}$ are non-negative. We devise active-set algorithms which allow us to relax the non-negativity constraints. We now show that when t_1(or t_2) $= 1$, $\boldsymbol{A}$ (or $\boldsymbol{Y}$) can be updated by our active-set NNQP algorithm described in Algorithm 3 and Section 4.6; when t_1(or t_2) $= 0$, $\boldsymbol{A}$ (or $\boldsymbol{Y}$) can be updated by our active-set 1_1QP algorithm described in Algorithm 2 and Section 4.6.

If $t_2 = 1$, the objective in Equation (2.78) can be rewritten as:

$$\begin{aligned} f(\boldsymbol{Y}) &= \text{trace}(\frac{1}{2}\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{Y} + \frac{1}{2}\boldsymbol{D}^{\mathrm{T}}\boldsymbol{D} - \boldsymbol{D}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{Y} + \frac{\lambda_2}{2}\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{Y} + \lambda_1\boldsymbol{E}_2^{\mathrm{T}}\boldsymbol{Y}) \\ &= \text{trace}(\frac{1}{2}\boldsymbol{Y}^{\mathrm{T}}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} + \lambda_2\boldsymbol{I})\boldsymbol{Y} + (\lambda_1\boldsymbol{E}_2^{\mathrm{T}} - \boldsymbol{D}^{\mathrm{T}}\boldsymbol{A})\boldsymbol{Y} + \frac{1}{2}\boldsymbol{D}^{\mathrm{T}}\boldsymbol{D}) \\ &= \sum_{i=1}^{n} \frac{1}{2}\boldsymbol{y}_i^{\mathrm{T}}\boldsymbol{H}_2\boldsymbol{y}_i + \boldsymbol{g}_{(2)i}^{\mathrm{T}}\boldsymbol{y}_i + \frac{1}{2}\boldsymbol{d}_i^{\mathrm{T}}\boldsymbol{d}_i, \end{aligned} \tag{2.89}$$

where $\boldsymbol{H}_2 = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} + \lambda_2\boldsymbol{I}$, and $\boldsymbol{g}_{(2)i} = \lambda_1 - \boldsymbol{A}^{\mathrm{T}}\boldsymbol{d}_i$ and $\boldsymbol{G}_{(2)} = \lambda_1 - \boldsymbol{A}^{\mathrm{T}}\boldsymbol{D}$. Therefore, we can see that updating non-negative $\boldsymbol{Y}$ is a multiple NNPQ problem. The parallel active-set algorithm for NNQP, proposed in Section 4.6, can be used to solve the problem in Equation (2.89).

If $t_2 = 0$, the objective of updating $\boldsymbol{Y}$ can be reformulated as:

$$\begin{aligned} f(\boldsymbol{Y}) &= \text{trace}(\frac{1}{2}\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{Y} + \frac{1}{2}\boldsymbol{D}^{\mathrm{T}}\boldsymbol{D} - \boldsymbol{D}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{Y} + \frac{\lambda_2}{2}\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{Y}) + \lambda_1\|\boldsymbol{Y}\|_1 \\ &= \text{trace}(\frac{1}{2}\boldsymbol{Y}^{\mathrm{T}}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} + \lambda_2\boldsymbol{I})\boldsymbol{Y} + (-\boldsymbol{D}^{\mathrm{T}}\boldsymbol{A})\boldsymbol{Y} + \frac{1}{2}\boldsymbol{D}^{\mathrm{T}}\boldsymbol{D}) + \lambda_1\|\boldsymbol{Y}\|_1 \\ &= \sum_{i=1}^{n} \frac{1}{2}\boldsymbol{y}_i^{\mathrm{T}}\boldsymbol{H}_2\boldsymbol{y}_i + \boldsymbol{g}_{(2)i}^{\mathrm{T}}\boldsymbol{y}_i + \lambda_1\|\boldsymbol{y}_i\|_1 + \frac{1}{2}\boldsymbol{d}_i^{\mathrm{T}}\boldsymbol{d}_i, \end{aligned} \tag{2.90}$$

where $\boldsymbol{H}_2 = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} + \lambda_2\boldsymbol{I}$, and $\boldsymbol{g}_{(2)i} = -\boldsymbol{A}^{\mathrm{T}}\boldsymbol{d}_i$ and $\boldsymbol{G}_{(2)} = -\boldsymbol{A}^{\mathrm{T}}\boldsymbol{D}$. This is a l_1QP problem which can be solved by our active-set l_1QP algorithm proposed in Section 4.6 and [35].

Similarly, if $t_1 = 1$, $f(\boldsymbol{A})$ in Equation (2.78) can be expressed as

$$\begin{aligned} f(\boldsymbol{A}) &= \text{trace}(\frac{1}{2}\boldsymbol{A}\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}} + \frac{1}{2}\boldsymbol{D}^{\mathrm{T}}\boldsymbol{D} - \boldsymbol{D}\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}} + \frac{\alpha_2}{2}\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}} + \alpha_1\boldsymbol{E}_1^{\mathrm{T}}\boldsymbol{A}) \\ &= \text{trace}(\frac{1}{2}\boldsymbol{A}(\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}} + \alpha_2\boldsymbol{I})\boldsymbol{A}^{\mathrm{T}} + (\alpha_1\boldsymbol{E}_1^{\mathrm{T}} - \boldsymbol{D}\boldsymbol{Y}^{\mathrm{T}})\boldsymbol{A}^{\mathrm{T}} + \frac{1}{2}\boldsymbol{D}\boldsymbol{D}^{\mathrm{T}}) \\ &= \sum_{i=1}^{m} \frac{1}{2}\boldsymbol{w}_i^{\mathrm{T}}\boldsymbol{H}_1\boldsymbol{w}_i + \boldsymbol{g}_{(1)i}^{\mathrm{T}}\boldsymbol{w}_i + \frac{1}{2}\boldsymbol{D}_{i,:}(\boldsymbol{D}^{\mathrm{T}})_{:,i}, \end{aligned} \tag{2.91}$$

where $\boldsymbol{W} = \boldsymbol{A}^{\mathrm{T}}$, $\boldsymbol{H}_1 = \boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}} + \alpha_2\boldsymbol{I}$, $\boldsymbol{g}_{(1)i} = \alpha_1 - \boldsymbol{Y}(\boldsymbol{D}^{\mathrm{T}})_{:,i}$ and $\boldsymbol{G}_{(1)} = \alpha_1 - \boldsymbol{Y}\boldsymbol{D}^{\mathrm{T}}$. Again, it can be seen that this problem is also a NNQP problem which can be solved by the active-set algorithm in Section 4.6 and [35].

If $t_1 = 0$, the objective of updating $\boldsymbol{A}$ can be written as

$$\begin{aligned} f(\boldsymbol{A}) &= \text{trace}(\frac{1}{2}\boldsymbol{A}\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}} + \frac{1}{2}\boldsymbol{D}^{\mathrm{T}}\boldsymbol{D} - \boldsymbol{D}\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}} + \frac{\alpha_2}{2}\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}}) + \alpha_1\|\boldsymbol{A}\|_1 \\ &= \text{trace}(\frac{1}{2}\boldsymbol{A}(\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}} + \alpha_2\boldsymbol{I})\boldsymbol{A}^{\mathrm{T}} + (-\boldsymbol{D}\boldsymbol{Y}^{\mathrm{T}})\boldsymbol{A}^{\mathrm{T}} + \frac{1}{2}\boldsymbol{D}\boldsymbol{D}^{\mathrm{T}}) + \alpha_1\|\boldsymbol{A}^{\mathrm{T}}\|_1 \\ &= \sum_{i=1}^{m} \frac{1}{2}\boldsymbol{w}_i^{\mathrm{T}}\boldsymbol{H}_1\boldsymbol{w}_i + \boldsymbol{g}_{(1)i}^{\mathrm{T}}\boldsymbol{w}_i + \alpha_1\|\boldsymbol{w}_i\|_1 + \frac{1}{2}\boldsymbol{D}_{i,:}(\boldsymbol{D}^{\mathrm{T}})_{:,i}, \end{aligned} \tag{2.92}$$

where $\boldsymbol{W} = \boldsymbol{A}^{\mathrm{T}}$, $\boldsymbol{H}_1 = \boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}} + \alpha_2\boldsymbol{I}$, $\boldsymbol{g}_{(1)i} = -\boldsymbol{Y}(\boldsymbol{D}^{\mathrm{T}})_{:,i}$ and $\boldsymbol{G}_{(1)} = -\boldsymbol{Y}\boldsymbol{D}^{\mathrm{T}}$. This is also a l_1QP problem that can

be solved by our active-set l_1QP algorithm proposed in Section 4.6 and [35].

Analytical Solutions and Kernelization

If $t_2 = 0$ and $\lambda_1 = 0$, from $\frac{\partial f(\boldsymbol{Y})}{\partial \boldsymbol{Y}} = 0$, $\boldsymbol{Y}$ can be updated analytically, that is,

$$\boldsymbol{Y} = (\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} + \lambda_2 \boldsymbol{I})^{-1}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{D} = \boldsymbol{A}^{\ddagger}\boldsymbol{D}. \tag{2.93}$$

From the previous section, we can see that only $\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}}$ and $\boldsymbol{Y}\boldsymbol{D}^{\mathrm{T}}$ are required to update $\boldsymbol{A}$. According to Equation (2.93), $\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}}$ and $\boldsymbol{Y}\boldsymbol{D}^{\mathrm{T}}$ can be expressed as

$$\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}} = \boldsymbol{A}^{\ddagger}\boldsymbol{D}\boldsymbol{D}^{\mathrm{T}}(\boldsymbol{A}^{\ddagger})^{\mathrm{T}}. \tag{2.94}$$

$$\boldsymbol{Y}\boldsymbol{D}^{\mathrm{T}} = \boldsymbol{A}^{\ddagger}\boldsymbol{D}\boldsymbol{D}^{\mathrm{T}}. \tag{2.95}$$

We can see that in this situation, updating $\boldsymbol{A}$ only requires the previous value of $\boldsymbol{A}$ and the inner products of the rows of $\boldsymbol{D}$.

Similarly, if $t_1 = 0$ and $\alpha_1 = 0$, $\boldsymbol{A}$ can be updated analytically, that is,

$$\boldsymbol{A} = \boldsymbol{D}\boldsymbol{Y}^{\ddagger}, \tag{2.96}$$

where $\boldsymbol{Y}^{\ddagger} = \boldsymbol{Y}^{\mathrm{T}}(\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}} + \alpha_2 \boldsymbol{I})^{-1}$. From the previous section, we know that updating $\boldsymbol{Y}$ only requires the inner products $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ and $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{D}$. They can be updated by the following equations:

$$\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} = (\boldsymbol{Y}^{\ddagger})^{\mathrm{T}}\boldsymbol{D}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{Y}^{\ddagger}. \tag{2.97}$$

$$\boldsymbol{A}^{\mathrm{T}}\boldsymbol{D} = (\boldsymbol{Y}^{\ddagger})^{\mathrm{T}}\boldsymbol{D}^{\mathrm{T}}\boldsymbol{D}. \tag{2.98}$$

Due to the analytical solution of $\boldsymbol{A}$, updating $\boldsymbol{Y}$ only requires the previous value of $\boldsymbol{Y}$ and the inner products of columns of $\boldsymbol{D}$.

These analytical solutions have two advantages. First, the corresponding matrix can be easily updated without resorting to any numerical solver. Second, we can see that only the inner products are needed to update $\boldsymbol{Y}$ (or $\boldsymbol{A}$), when $\boldsymbol{A}$ (or $\boldsymbol{Y}$) can be analytically obtained. Using this property, we can obtain the kernel version of VSMF, which are described in the following. In sparse representation, at least one matrix among $\boldsymbol{A}$ and $\boldsymbol{Y}$ must be sparse. That is the analytical solutions in Equations (2.93) and (2.96) can not be used simultaneously. In practice, if each column of the training data $\boldsymbol{D}$ is the object to be mapped in high-dimensional feature space, we can analytically update $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ (or the corresponding kernel version $k(\boldsymbol{A},\boldsymbol{A}) = (\phi(\boldsymbol{A}))^{\mathrm{T}}\phi(\boldsymbol{A})$ where $k(\cdot,\cdot)$ is a kernel function corresponding to $\phi(\cdot)$) and $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{D}$ (or $k(\boldsymbol{A},\boldsymbol{D}) = (\phi(\boldsymbol{A}))^{\mathrm{T}}\phi(\boldsymbol{D})$), and then update $\boldsymbol{Y}$ via a numerical solver described in the previous section. This leads to the kernel sparse representation

proposed in [35]. Alternatively, if each row of $\boldsymbol{D}$ is the object to be mapped in high-dimensional feature space, $\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}}$ and $\boldsymbol{Y}\boldsymbol{D}^{\mathrm{T}}$ should be updated analytically, then $\boldsymbol{A}$ is updated by a solver given in the previous section. This leads to an alternative kernel sparse representation model.

7.3 Computational Experiment

Sparse matrix factorization has a wide ranges of applications in biological data analysis. Technically speaking, these applications are based on clustering, biclustering, feature extraction, classification, and feature selection. In this study, we give three examples to show that promising performance can be obtained by VSMF for feature extraction, feature selection, and biological process identification. For other applications of NMF, please refer to [12].

Feature Extraction and Classification

NMF is a successful feature extraction method in bioinformatics [38]. Dimensionality reduction including feature extraction and feature selection is an important step in classification. We compared the performance of our VSMF (for feature extraction) with NMF on a popular microarray gene expression data – Colon [39]. This data set has 2000 genes (features) and 62 samples. There are two classes in this data set. Each sample is normalized to have unit l_2-norm. We employed 4-fold cross-validation to split the whole data into training and test sets. For each split, features were extracted by NMF or VSMF from the training set. The *nearest neighbor* (NN) classifier was used to predict the class labels of the test set. 4-fold cross-validation was repeated for 20 times. We initialized $k = 8$, thus the actual value of k, after calling VSMF, should be less than or equal to 8. *Radial basis function* (RBF) is used in the kernel VSMF. We set the kernel parameter $\sigma = 2^0$. The mean accuracy, *standard deviation* (STD), computing time, and parameter setting are given in Table 2.5. The standard NMF obtained a mean accuracy of 0.7645, while the linear VSMF yielded 0.7919. The highest accuracy, 0.7944, is obtained by the kernel VSMF. The kernel VSMF only took 1.3346 seconds, which is faster than the linear VSMF and NMF, because the analytical solution of $\boldsymbol{A}$ can be computed for kernel VSMF. We treat this comparison as a demonstration that tuning the parameters of VSMF may obtain better accuracy than NMF. VSMF can be used for many other types of high-throughput data such as copy number profiles and mass spectrometry data.

Table 2.5: Classification performance of VSMF compared to the standard NMF. The time is measure by stopwatch timer (the `tic` and `toc` functions in MATLAB) in seconds.

Method	Accuracy (STD)	Time	Parameters
NN	0.7742(0.0260)	0.0137	-
NMF+NN	0.7645(0.0344)	4.3310	-
Linear VSMF+NN	0.7919(0.0353)	3.1868	$\alpha_2 = 2^{-3}, \lambda_1 = 2^{-6}, t_1 = t_2 = 1$
Kernel VSMF+NN	0.7944(0.0438)	1.3346	$\alpha_2 = 2^{-3}, \lambda_1 = 2^{-6}, t_1 = t_2 = 1, \sigma = 2^0$

Feature Selection

VSMF can be applied to select relevant features. The basic idea is to make the basis vectors sparse, and then select features that vary dramatically among the basis vectors. In our current study of gene selection (genes are features), we use the following strategy on the sparse basis matrix $\boldsymbol{A}$. For the i-th row (that is the i-th gene), We denote $\boldsymbol{g}_i = \boldsymbol{A}_{i,:}$. If the maximum value in $\boldsymbol{g}_i$ is $\theta = 10^4$ times greater than the remaining values in $\boldsymbol{g}_i$, then we select this gene, otherwise we discard it. We tested this VSMF-based feature selection method on a microarray breast tumor data set which have 13582 genes and 158 samples from five classes [29]. The data were normalized so that each gene has mean zero and standard deviation (STD) one. We used the following parameters of VSMF: $\alpha_1 = 2^4$, $\alpha_2 = 2^0$, $\lambda_1 = 0$, $\lambda_2 = 2^0$, $t_1 = 0$, and $t_2 = 1$. The value of k was initialized to 5. The selected genes were validated by classification performance. We employed 20 runs of four-fold cross-validation. For each split of training and test sets, genes were selected using the training set. On the reduced training set, a linear SVM was trained in order to predict the class labels of the corresponding test set. When using all genes to training SVM, we obtained a mean accuracy of 0.8250 with STD 0.0201. When applying the VSMF-based gene selection, we achieved a mean accuracy of 0.8271 with STD 0.0174. We can see that SVM using our gene selection strategy can obtain similar performance with that of using all genes.

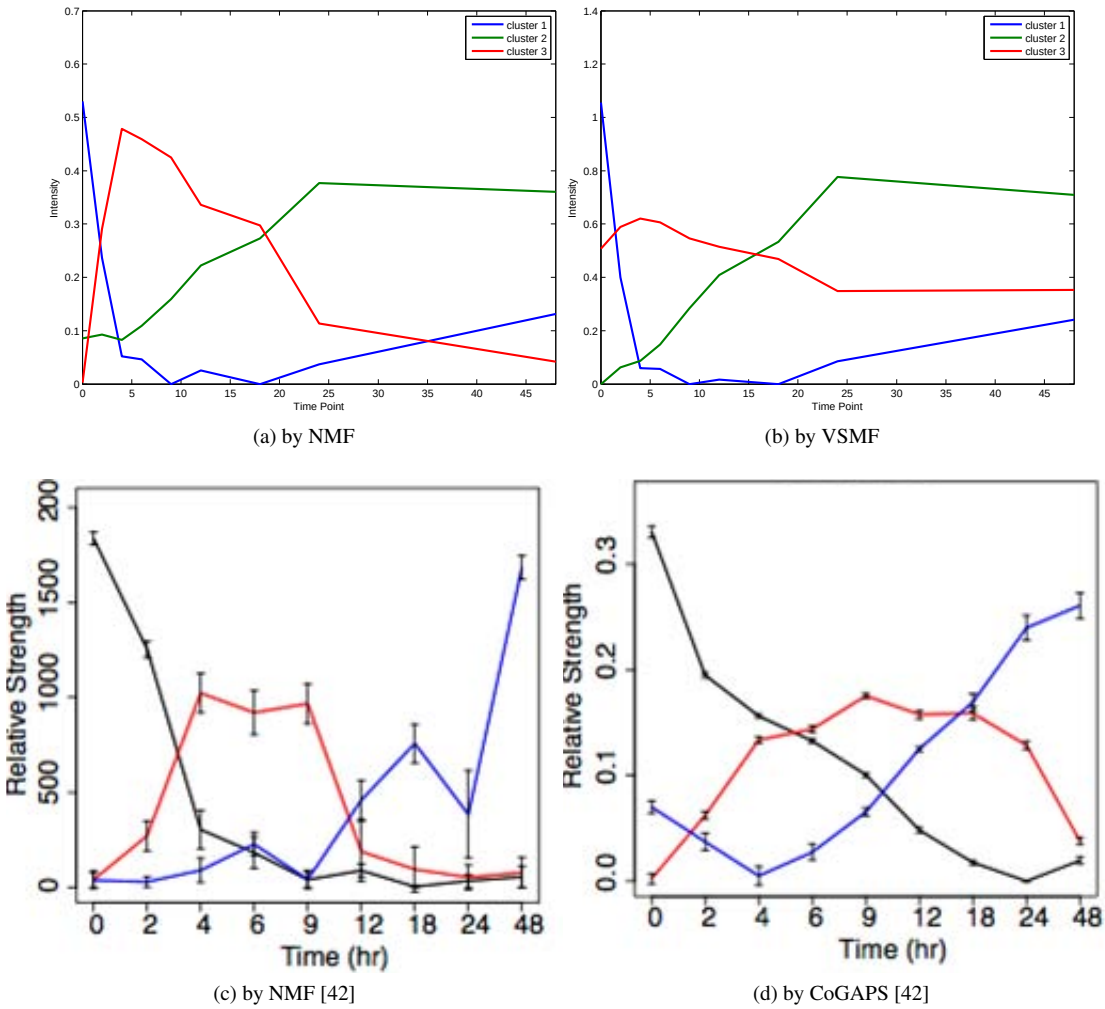
7.4 Biological Process Identification

NMF has been applied on either gene-sample microarray data or time-series microarray data in order to identify potential biological processes [40, 34, 41, 42]. A biological process here means the genotypic pattern of a pathway. In our experiment, we run our VSMF on the *Gastrointestinal stromal tumor* (GIST) time-series data [42] to show that VSMF can smooth biological processes compared with the result obtained by the standard NMF. This data set was obtained after the treatment of imatinib mesylate. It has 1336 genes and 9 time points. Each gene time-series is normalized to have unit l_2-norm. The smoothness is controlled by parameter α_2. We set the parameters of VSMF to $\alpha_2 = 2^{-2}$, $\lambda_1 = 2^{-8}$, $\alpha_1 = \lambda_2 = 0$, and $t_1 = t_2 = 1$. The number of factors k was set to 3. The basis vectors of NMF and VSMF are shown in Figures 2.5a and 2.5b, respectively. The results obtained by the NMF and *coordinated gene activity in pattern sets* (CoGAPS, a variant of NMF optimized by Markov chain Monte Carlo sampling) of Ochs *et al.* are shown in Figures 2.5c and 2.5d. When comparing the plots in Figure 2.5, please note that the horizontal axis on the top are different from that at the bottom. We can see that all methods are able to reconstruct the falling, rising, and transient patterns identified in [42]. The patterns obtained by VSMF and CoGAPS are smoother than that of the standard NMF.

8 Supervised Dictionary Learning

When applying sparse representation to classifying a data set with a large number of classes and samples, we have the following challenges:

1. As instance-based learning, the sparse coding methods without dictionary learning are able to classify



(a) by NMF (b) by VSMF

(c) by NMF [42] (d) by CoGAPS [42]

Figure 2.5: The biological processes identified by our implementations of the standard NMF (a) and VSMF (b), and by Ochs *et al.*'s implementations of the standard NMF (c) and CoGAPS (d) [42].

complex data, but become very slow as the number of samples increases dramatically.

2. The unsupervised dictionary learning learns a generative model and maintains a small dictionary, however it does not learn well the representations of complex multi-class data.

Taking the advantages of both sparse coding and dictionary learning, we propose the concept of *sub-dictionary learning*. The work of *meta-sample based sparse representation classification* (MSRC) [43] is in fact a specific case of sub-dictionary learning. MSRC employs SVD to learn the sub-dictionaries. The sparse coding of a new sample is obtained through solving Equation (2.10). After that, the NS rule is used to predict the class label. Sound performance of MSRC was reported over microarray data. NMF was also tried to learn

sub-dictionaries, but it was claimed, in [43], that its performance is worse than that of using SVD.

We find the sub-dictionary learning principle has the following advantages: i) this is eager learning, hence is applicable in real-time prediction; ii) dictionary learning over data of large sample size and many classes can be solved through a divide-and-conquer scheme; iii) the learning on each class is independent from others, therefore the classifier is easy to upgrade when there are some new training samples from some of the classes; and iv) this is actually supervised learning because sub-dictionaries are learned for all classes separately.

NMF has been used as a sub-dictionary learning method by MSRC. However, it is unclear if semi-NMF has been used by MSRC in the case of mixed-sign data. Also, if any NMF is used as dictionary learning approach, it is not suitable to pursue the l_1LS sparse coding. This is because a new sample is supposed to be a non-negative linear combination of the dictionary atoms obtained by a NMF, but the coefficient vector obtained by l_1 regularization may contain negative numbers. Thus it makes more sense if NNLS sparse coding is used after the NMF based sub-dictionary learning.

Generally speaking, the sparse coding model must match the dictionary learning model in an implementation of the sub-dictionary learning principle. In this section, we give such an implementation based on non-negative dictionary learning and non-negative sparse coding. This method is called *meta-sample based NNLS* (MNNLS) classification approach, and is described in Algorithm 6. A *meta-sample* is defined as a column of the basis matrices obtained via (sub-)dictionary learning. In the algorithm, the meta-samples obtained by NMF from the i-th class are labeled to belong to this class. NMF can be solved by the generic alternating framework described in Section 6.1 or the versatile sparse matrix factorization framework which is discussed in Section 7.

Algorithm 6 *MNNLS Classification Approach*

Input: $\boldsymbol{D}_{m\times n}$: training data, $\boldsymbol{c}$: class labels of n training samples, $\boldsymbol{B}_{m\times p}$: new data, $\boldsymbol{k}$: size of sub-dictionaries
Output: $\boldsymbol{p}$: the predicted class labels of the p new samples

Training Step:

1. Normalize each training sample to have unit l_2 norm;
2. For any class, learn the sub-dictionary through NMF ($\boldsymbol{D}_{i+} \approx \boldsymbol{A}_{i+}\boldsymbol{Y}_{i+}$ where $\boldsymbol{D}_i$ is the training samples of the i-th class in input space and $\boldsymbol{Y}_{i+}$ is their representation in feature space) if the data are non-negative, or semi-NMF ($\boldsymbol{D}_i \approx \boldsymbol{A}_i\boldsymbol{Y}_{i+}$) if the data are of mixed signs;
3. Concatenate the sub-dictionaries into a holistic dictionary via $\boldsymbol{A} = [\boldsymbol{A}_1, \cdots, \boldsymbol{A}_C]$;

Prediction Step:

1. Normalize each new sample to have unit l_2 norm;
2. Solve the NNLS $\min_{\boldsymbol{X}} \frac{1}{2}\|\boldsymbol{B} - \boldsymbol{A}\boldsymbol{X}\|_F^2$ s.t. $\boldsymbol{X} \geq 0$;
3. For each column of $\boldsymbol{X}$, apply nearest-subspace rule to predict the corresponding class label;

8.1 Computational Experiments

We examined the performance of MNNLS on Dawany's data set [31] which has 5456 microarray gene expression samples distributed in 26 classes (see Section 5.2). We compared MNNLS with NNLS (sparse coding method), NMF (unsupervised dictionary learning), MSRC-SVD, and MSRC-NMF (proposed in [43]). We conducted 10-fold cross-validation to split the data into training and test sets. All the methods are trained on the same training sets. The best mean prediction accuracies, computing time, and corresponding parameters are given in Table 2.6.

Table 2.6: The performance of MNNLS.

Method	Accuracy	Time (second)	Parameter
NNLS	0.9762	998	linear kernel
MNNLS	0.9622	618	$k_i = 8$
MSRC-SVD	0.9617	8944	$k_i = 8, \lambda = 2^{-10}$
MSRC-NMF	0.7654	341	$k_i = 8, \lambda = 2^{-3}$
NMF	0.1391	2.2730e+005	$k = 26$

From the result, first of all, we can see that MNNLS has similar accuracy with the sparse coding method NNLS. Meanwhile, MNNLS took less time than NNLS. Second, the accuracy of MSRC-NMF is very poor, compared to that of MNNLS. This corroborates that the sparse coding method, in a sub-dictionary learning, needs to match with the sub-dictionary learning method carefully. Third, the unsupervised method NMF obtained much worse accuracy, and spent much more time than MNNLS. This confirms that unsupervised dictionary learning may not capture the key patterns of complicated data.

9 Local NNLS Method

In this section, we propose two variants of the NNLS method to classify high dimensional microarray data. Our methods combine the advantages of NMF, local learning, transductive learning, and ensemble learning.

9.1 Local NNLS Classifier

Interesting but complicated patterns may be hidden within microarray data. Local learners may approximate this complexity by learning in local regions. In order to emphasize the advantages of local learning, we propose a generalized local version of NNLS which is described in Algorithm 7. Please note that, the NNLS classifier given in Section 3 can be viewed a special case of this local approach, if we assume the cluster number is one. Given $\boldsymbol{X}$ containing labeled training samples in columns, the classification task is to categorize the unknown unlabeled samples $\boldsymbol{S}$. Our idea of solving this task is the following. First, the union of $\boldsymbol{X}$ and $\boldsymbol{S}$ is clustered by NMF (if it is non-negative) or semi-NMF (if it is of mixed signs). Then, for each cluster, if there is no unlabeled sample in this cluster, simply skip this iteration; if this cluster does not contain any

training sample, a NNLS classifier is applied over the whole training set $\boldsymbol{X}$ to classify the unlabeled samples in this cluster; if there are both training samples and unlabeled samples in this cluster, then a NNLS classifier learns on these training samples and predict the classes of these unlabeled samples. This approach is also transductive as unlabeled samples are involved in the partitioning phase. This method is named *local NNLS* (LNNLS) classification approach. We denote it as LNNLS-MAX, LNNLS-KNN, and LNNLS-NS for MAX (or NN), K-NN, and NS rules, respectively.

Profile SVM (PSVM) [44] is a local SVM algorithm. The main differences between this approach and PSVM are that i) PSVM has to conduct a supervised partitioning as the learning of a linear SVM in a subgroup require (balanced) samples from both classes, while LNNLS uses unsupervised clustering which neither deteriorate the natural structure of the data, nor require classifiers for some homogeneous clusters (if all the training samples in a cluster are from the same class, then the cluster is *homogeneous*); and ii) PSVM is a binary approach, and hence, computationally costly in the case of multiple classes, whereas LNNLS is appropriate for multi-class data.

Due to the relatively small sample size of microarray data, some readers might be concerned that learning in local regions may worsen the learning performance since the size of a cluster is smaller. We address this concern with the following three points. First, the optimization of NMF and NNLS needs only the inner products of the samples rather than the original high dimensional vectors, and thus, the classification is in fact dimension-free. Replacing the inner products with kernel matrices, we can obtain kernel NMF and kernel NNLS, though we have only investigated linear kernel so far. Second, our LNNLS method does not simply cluster and classify. Applying a classifier in a non-homogeneous cluster (i.e., a cluster containing labeled samples from different classes) is the final step of prediction. For homogeneous clusters, that is when all labeled samples are from the same class c, we can directly assign the class c as the label of the unlabeled samples contained in such clusters. For clusters in which all samples are unlabeled, all training samples will be required for classifying the unlabeled samples. Third, whether the sample size is sufficient or not is a statistical issue, it also depends on the biological and experimental complexity (please see [45] and [46] and references therein). Despite their small sample size, microarray data may still contain enough information needed for discovering the hidden patterns with a statistical learning model. In Section 9.2 we will empirically show that the minimum number of training samples required by NNLS to obtain a significant accuracy is generally very small on microarray data.

9.2 Repetitive LNNLS Classifier

The performance of the LNNLS classifier depends on the clustering algorithm (NMF or semi-NMF). Due to the non-convexity of NMF optimization and to the random initializations, different executions of NMF may result in different factors, and therefore we may obtain different clustering results. We thus propose the *repetitive LNNLS* (RLNNLS) classification approach, in Algorithm 8, in which we perform LNNLS learning on the data *maxR* times. The final class labels of the unlabeled samples are then decided through a voting process on the decisions returned by the *maxR* LNNLS classifiers. The *majority rule* is used as our voting method. Thus, this technique falls in the domain of ensemble learning [47]. From now on, RLNNLS

Algorithm 7 *LNNLS Classifier*

Input: $\boldsymbol{X}_{m\times n}$: training set with m features and n samples
$\boldsymbol{c}$: class labels of the training samples
$\boldsymbol{S}_{m\times p}$: p unknown samples without labels
Clust_Method: clustering method; *NMF*, *semi-NMF*
Output: $\boldsymbol{p}$: predicted class labels of the p unknown samples

$[\mathbf{X}_i, \mathbf{S}_i]_{i=1}^{i=k} \leftarrow Clustering([\boldsymbol{X}, \boldsymbol{S}], Clust\_Method)$;
$\{\mathbf{X}_i$: *set of labeled samples in i-th cluster*$\}$
$\{\mathbf{S}_i$: *set of unlabeled samples in i-th cluster*$\}$

for $i \leftarrow 1$ **to** k **do**
  if $\boldsymbol{S}_i = \emptyset$ **then**
    Continue;
  end if

  if $\boldsymbol{X}_i = \emptyset$ **then**
    $\boldsymbol{p}_i \leftarrow NNLS(\boldsymbol{X}, \boldsymbol{c}, \boldsymbol{S}_i)$;
      $\{\boldsymbol{p}_i$: *predicted labels of samples in* $\boldsymbol{S}_i\}$
    Continue;
  end if

  if $Homogeneous(\boldsymbol{X}_i)$ **then**
    $\boldsymbol{p}_i \leftarrow \boldsymbol{c}_i$; $\{\boldsymbol{c}_i$ *is the unique class label appearing in* $\boldsymbol{X}_i\}$
    Continue;
  end if
  $\boldsymbol{p}_i \leftarrow NNLS(\boldsymbol{X}_i, \boldsymbol{c}_i, \boldsymbol{S}_i)$;
end for

return $\boldsymbol{p}$.

coupled with MAX, K-NN, and NS rules are denoted as *RLNNLS-MAX*, *RLNNLS-KNN*, and *RLNNLS-NS*, respectively.

Algorithm 8 *RLNNLS Classifier*

Input: $\boldsymbol{X}_{m\times n}$: training set with m features and n samples
$\boldsymbol{c}$: class labels of the training samples
$\boldsymbol{S}_{m\times p}$: p unknown samples without labels
Output: $\boldsymbol{p}$: predicted class labels of the p unknown samples

for $r \leftarrow 1$ **to** *maxR* **do**
  $\mathbf{p}_r \leftarrow LNNLS(\boldsymbol{X}, \boldsymbol{c}, \boldsymbol{S})$; $\{\mathbf{p}_r$: *predicted labels at the r-th iteration*$\}$
end for

return $\boldsymbol{p} \leftarrow Majority\_Vote(\mathbf{p}_1, \ldots, \mathbf{p}_{maxR})$.

Experiment

Table 2.7: High dimensional microarray data sets.

Data	+/-	#Classes	#Features	#Samples
Adenoma[48]	±	2	7457	18+18=36
Breast[49]	+	2	24481	44+34=74
Colon[39]	+	2	2000	40+22=62
DLBCL-NIH[50]	±	2	7399	102+138=240
Leukemia[51]	±	2	7129	47+25=72
Lung[52]	±	2	7129	10+86=96
Medulloblastoma[53][54]	+	2	5893	25+9=34
Prostate[55]	±	2	12600	59+77=136
ALL[56]	±	6	12625	15+27+64+20+43+79=248
ALLAML[51][54]	+	3	5000	19+8+11=38
Breast5[29]	±	5	13582	39+22+53+31+13=158
CNS[57]	±	5	7129	10+10+10+4+8=42
MLL[58]	±	3	12582	24+20+28=72
SRBCT[59]	+	4	2308	23+8+12+20=63
Bladder[60]	±	2	2143	32+16=48
BreastBerkeley[61]	±	2	2149	72+69=141
Melanoma[62]	±	2	3649	35+43=78

In order to investigate the classification performance of our family of NNLS classifiers for high dimensional biological data, we ran it on 14 microarray gene expression data sets (including 8 binary-class and 6 multi-class data sets) and 3 binary-class array-based comparative genomic hybridization (aCGH) data sets. The aCGH technique is used to identify DNA copy numbers. These data are summarized in Table 2.7. The aCGH data sets are in the last block of the table. The numbers of features vary from 2000 to 24481 among these 17 data sets. The numbers of samples vary from 34 to 248. The gene expression intensities are naturally non-negative. However, due to preprocessing when producing the data, some data have negative components. 5 out of these 14 data sets, are non-negative. The aCGH data are presented as $\log_2$-ratio, therefore have both positive and negative values. In this study, for non-negative data, NMF is used to cluster the data; otherwise, semi-NMF is employed to do the same task. The optimal number of clusters was obtained by linear search, though a model selection method proposed in [54] is widely used. This is because we observed that the optimal number of clusters suggested by that model selection method does not always result in the highest classification accuracy. We used linear kernel in our NNLS family. We compared our NNLS family with three classical local learners, namely, *k*-NN, SVM with RBF kernel, and LSVM. We implemented all these methods in MATLAB. The recently proposed *sparse PLS discriminant analysis* (SPLSDA) and *sparse generalized PLS* (SGPLS) [63] were also compared with our methods. We use the SPLS package (version 2.1-1) in R which is available from [63]. Parameters of all methods were selected by linear or grid search. 4-fold cross-validation was used to split a whole data set into labeled training set and unlabeled test set. We defined the *accuracy* of a given classifier as the ratio of the number of correctly predicted test samples to the total number of test samples. For each data set, four-fold cross-validation was rerun 20 times, and the averages and standard deviations over the 20 runs were computed.

Figure 2.6 shows the results on six data sets: Breast, Colon, Prostate, ALL, Breast5, and BreastBerkeley.

SGPLS fails on the ALL data set which has a very large number of features. LSVM gives better performance compared with SVM on Breast, ALL, and BreastBerkeley, and gives very poor performance on Colon and Prostate. From all six groups of results, it can be seen that the medium performance was obtained by k-NN. It can also be seen that our NNLS classifiers generally perform very well. We can also see that NNLS-NS has better accuracies than NNLS-MAX on Prostate and Breast5. We can observe that LNNLS-NS obtained better results than NNLS-NS. Furthermore, the RLNNLS-NS classifier has better accuracies than LNNLS-NS on Breast, Prostate, and ALL; meanwhile comparable results were obtained by both of them over the remaining three data sets.

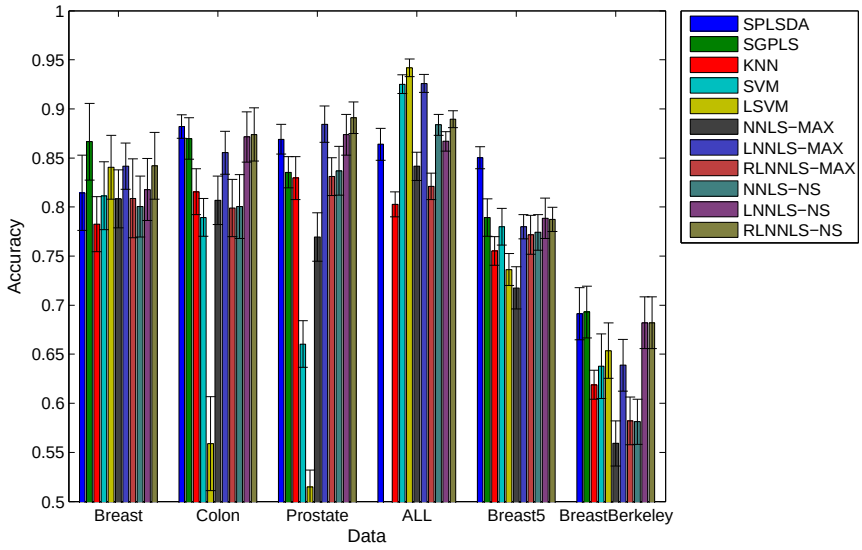


Figure 2.6: Classification performance on six data sets. This a color figure, thus may affect the readability if printed in grayscale. The order of the bars, from left to right, in the figure, is the same as these from top to bottom in the legend.

Statistical Comparison

We used the Friedman test with the post-hoc Nemenyi test to further statistically compare all the classifiers over their mean accuracies on the 17 data sets. The Friedman test is a non-parametric approach which compares multiple classifiers on multiple data sets [64]. It has been recommended by [64], since it is simple, safe, and robust. In the Friedman test, the classifiers are ranked for each data set. The classifier obtained the best performance has rank 1, and the second best classifier gets rank 2, and so on. The rank of ties is their rank average. For example the 3-rd and 4-th classifiers have the same performance on a data set, then both of them get rank 3.5. The average rank of each classifier are then obtained over all data sets. The statistic of the

test is a function of the average ranks of all classifiers:

$$F_F = \frac{(N-1)\xi_F^2}{N(k-1)-\chi_F^2}, \tag{2.99}$$

where N is the number data sets, k is the number of classifiers, and χ_F^2 is the Friedman statistic, and is defined as

$$\chi_F^2 = \frac{12N}{k(k+1)}\Big(\sum_{j=1}^{k} R_j^2 - \frac{k(k+1)^2}{4}\Big), \tag{2.100}$$

where R_j is the average rank of the j-th classifier. Statistic F_F follows F-distribution with degrees of freedom $(k-1,(k-1)(N-1))$. The null hypothesis of the Friedman test is that all classifiers are equivalent in term of error rate. Once it is rejected, the Nemenyi test is then used to find the difference among the classifiers. In the Nemenyi test, the *crucial difference* (CD) is computed as

$$CD = q_\alpha\sqrt{\frac{k(k+1)}{6N}}, \tag{2.101}$$

where α is the significant level, and q_α is the upper crucial value (upper quantile) of the Studentized range distribution (with infinite degree of freedom) divided by $\sqrt{2}$. During the Nemenyi test, if the distance between the average ranks of a pair of classifiers are larger than the CD, then we say that they are significantly different. The significance level was set to $\alpha = 0.05$ in our experiment. The null hypothesis was rejected in our test, and thus the post-hoc Nemenyi test was conducted. The graphical presentation of the Nemenyi test can be found in Figure 2.7 . According to the test results, we have the following interpretations which are consistent with our above observations from Figure 2.6.

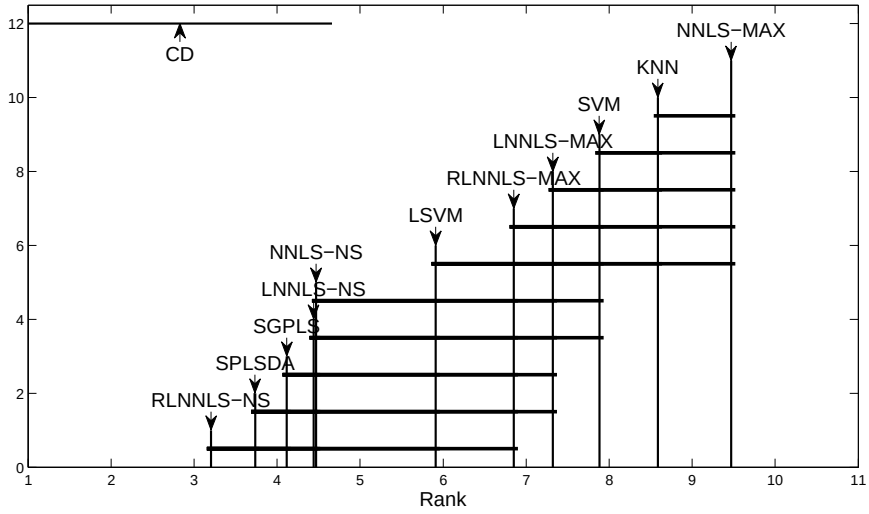


Figure 2.7: Crucial difference diagram of the Nemenyi test ($\alpha = 0.05$).

NNLS using NS rule is significantly better than the one using MAX rule. NNLS-NS is significantly better than k-NN. There is a large difference between the average ranks of NNLS-NS and SVM, though the difference is not significant at the current significance level. If we increase α slightly, then NNLS-NS and SVM will be significantly different. The average rank of LNNLS is not worse than NNLS. RLNNLS-NS obtained the best average rank. It is significantly different from LNNLS-MAX, SVM, k-NN, and NNLS-MAX. Finally, the average ranks of SPLSDA and SGPLS are between those of RLNNLS-NS and LNNLS, and current test does not find significant difference among them.

Computing Time

The averaged elapsed execution time in seconds of each method was also recorded as a measure of performance. All experiments were performed on an Intel machine (Core TM i7, 2.93 GHz, CPU with 8 GB RAM, with 64-bit Windows 7 Professional operation system). All methods, except the SPLS methods, were implemented in the language MATLAB, 64-bit version 7.11. The SPLS methods was implemented in the language R, 64-bit version 2.12. The computing time of LNNLS and RLNNLS does not include the time of clustering, because clustering can be done only once and then the results can be saved in memory or hard drive. During the iterations of cross-validation, the same clustering results can be reused through simply changing the roles (training or test) of the samples. In real-world applications, to predict the class labels of new but unlabeled samples, the computing time may increase since clustering must be redone. *Online* updating methods can be investigated to reduce the computing time of NMF clustering; see [65] for an example of an online NMF. Figure 2.8 shows the computing time of the methods on the six data sets; $\log_2$ *times* are given for a better visual comparison.

First, it can be clearly seen that our NNLS approaches are much more efficient than the PLS methods over large or multi-class data. For instance on ALL, the computing time of SGPLS is as 10 and 7 times as NNLS-NS and RLNNLS-NS, respectively. Second, our inductive and local NNLS methods are also consistently faster than local SVM. Local SVM becomes computationally costly when the number of classes is large. For example over Breast5 and ALL, it performs slower than our RLNNLS methods. One of the reason for the efficiency of our methods, compared to PLS and LSVM methods, is because our techniques are naturally multi-class methods, while the supervised PLS and LSVM are binary models which have to use one-against-all, one-against-one, or other strategies for multi-class task. Therefore their computing time is significantly increased in the multi-class case.

Estimation of Data-Size Requirement

We estimated the minimum data size (that is, the number of training samples) required to obtain significant accuracy, in order to explore the practical reason for the high performance of our approaches. We implemented the permutation-test-based method proposed in [45]. We set the significance level to $\alpha = 0.05$ in these experiments. The minimum data size of NNLS-NS and SVM for each data set is listed in Table 2.8. In the experiments above, all classifiers obtained poor performance on DLBCL-NIH. Through investigating the data-size requirements, we found that the sample size provided in this data set was not sufficient to obtain

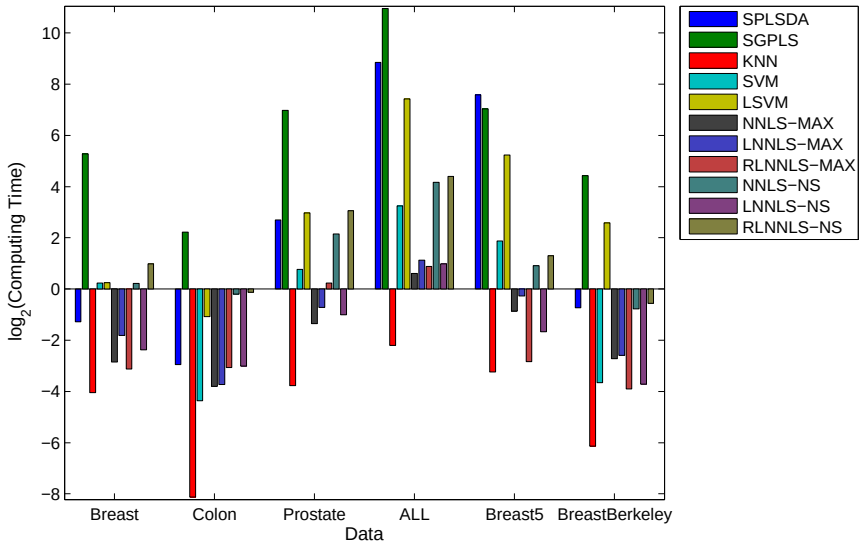


Figure 2.8: Computing time on six data sets. This a color figure, thus may affect the readability if printed in grayscale. The order of the bars, from left to right, in the figure, is the same as these from top to bottom in the legend.

a significant accuracy. In Table 2.8, we can see that NNLS-NS generally needs fewer training samples in order to achieve significant accuracy. This also is a possible reason of why LNNLS methods obtained good accuracies, in practice, as they perform prediction via clustering.

Table 2.8: Minimum data size required to obtain significant accuracy ($\alpha = 0.05$). NNLS-NS method only requires a very small number of training samples in order to obtain significant accuracy.

Data	NNLS-NS	SVM
Adenoma	6	2
Breast	10	12
Colon	8	12
DLBCL-NIH	-	-
Leukemia	6	12
Lung	15	11
Medulloblastoma	12	2
Prostate	9	11
ALL	6	10
ALLAML	5	10
Breast5	5	11
CNS	5	14
MLL	4	18
SRBCT	4	10
Bladder	10	8
BreastBerkeley	20	11
Melanoma	6	4

10 Conclusions

In this chapter, we proposed the Bayesian sparse coding and dictionary learning models. We intensively investigated three sparse coding models: the l_1-regularized model, the non-negative model, and l_1-regularized non-negative model. We revealed that these models can be kernelized. We reviewed main sparse coding optimization methods, and proposed our own active-set method and decomposition method. We proposed a generic network of kernel dictionary learning. We then proposed a more general framework, named versatile sparse matrix factorization. The classification, feature extraction, feature selection techniques based on sparse representation for high-dimensional biological data were investigated. We proposed two novel classification techniques – the sub-dictionary learning and local sparse coding for complicated biological data.

There are some interesting future researches. We just mentioned a few in the following. First, a novel supervised dictionary learning can be devised by combining Bayesian dictionary learning and Bayesian regression. Second, it would be interesting to investigate supervised kernel dictionary learning. Third, optimization methods based on Markov chain Monte Carlo sampling can be implemented for the sparse representation models.

Chapter 3

The Non-Negative Matrix Factorization and Sparse Representation Toolboxes

1 Introduction

Open-source packages are very important for both researchers in the machine learning and bioinformatics communities. From receiving feedbacks and suggestions from users, the developers can improve their implementations, and add new functionalities. By using these packages, researchers in the communities enjoy the convenience of one-stop implementation of most important methods of a computational theory.

We have already implemented two toolboxes including *non-negative matrix factorization* (NMF) toolbox and *sparse representation* (SR) toolbox. NMF and SR have evolved relatively independently in the machine learning field, though NMF is essentially a non-negative sparse representation model. NMF is intensively studied in both machine learning and bioinformatics, while the general SR is still not popular in bioinformatics. For this reason, we separate the implementations of NMF and SR. Since NMF is a special case of sparse representation, this is almost unavoidable that there are some overlapping in the both implementations. However, we tried our effort to minimize this overlapping. In the following sections, we first introduce our NMF toolbox, and then the SR toolbox. Since the importance of NMF, we also briefly describe important variants of NMF in the following section.

2 The Non-Negative Matrix Factorization Toolbox

2.1 Background

Non-negative matrix factorization (NMF) is a matrix decomposition approach which decomposes a non-negative matrix into two low-rank non-negative matrices [13]. It has been successfully applied in the mining of biological data.

For example, The authors of [54] and [34] used NMF as a clustering method in order to discover the metagenes (i.e., groups of similarly behaving genes) and interesting molecular patterns. The authors of [66] applied *non-smooth NMF* (NS-NMF) for the biclustering of gene expression data. *Least-squares NMF* (LS-NMF) was proposed to take into account the uncertainty of the information present in gene expression data [67]. The authors of [68] proposed kernel NMF for reducing dimensions of gene expression data.

Many authors indeed provide their respective NMF implementations along with their publications so that the interested community can use them to perform the same data mining tasks respectively discussed in those publications. However, there exists at least three issues that prevent NMF methods from being used by the much larger community of researchers and practitioners in the data mining, biological, health, medical, and bioinformatics areas. First, these NMF softwares are implemented in diverse programming languages, such as R, MATLAB, C++, and Java, and usually only one optimization algorithm is provided in their implementations. It is inconvenient for many researchers who want to choose a suitable NMF method or mining task for their data, among the many different implementations, which are realized in different languages with different mining tasks, control parameters, or criteria. Second, some papers only provide NMF optimization algorithms at a basic level rather than a data mining implementation at a higher level. For instance, it becomes hard for a biologist to fully investigate and understand his/her data when performing clustering or biclustering of his data and then visualize the results; because it should not be necessary for him/her to implement these three data mining methods based on a basic NMF. Third, the existing NMF implementations are application-specific, and thus, there exists no systematic NMF package for performing data mining tasks on biological data.

There currently exists NMF toolboxes, however, none of them addresses the above three issues altogether. *NMFLAB* [69] is MATLAB toolbox for signal and image processing which provides a user-friendly interface to load and process input data, and then save the results. It includes a variety of optimization algorithms such as multiplicative rules, exponentiated gradient, projected gradient, conjugate gradient, and quasi-Newton methods. It also provide methods for visualizing the data signals and their components, but does not provide any data mining functionality. Other NMF approaches such as semi-NMF and kernel NMF are not implemented within this package. *NMF:DTU Toolbox* [70] is a MATLAB toolbox with no data mining functionalities. It includes only five NMF optimization algorithms, such as multiplicative rules, projected gradient, probabilistic NMF, alternating least squares, and alternating least squares with *optimal brain surgery* (OBS) method. *NMFN: Non-negative Matrix Factorization* [71] is an R package similar to *NMF:DTU* but with few more algorithms. *NMF: Algorithms and framework for Nonnegative Matrix Factorization* [72] is another R package which implements several algorithms and allows parallel computations but no data mining functionalities. *Text to Matrix Generator (TMG)* is a MATLAB toolbox for text mining only. The authors of [73] provides a NMF plug-in for BRB-ArrayTools. This plug-in only implements the standard NMF and semi-NMF and for clustering gene expression profiles only. *Coordinated Gene Activity in Pattern Sets* (CoGAPS) [74] is a new package implemented in C++ with R interface. In this package, the *Bayesian decomposition* (BD) algorithm is implemented and used in place of the NMF method for factorizing a matrix. Statistical methods are also provided for the inference of biological processes. CoGAPS can give more precise results than NMF methods [42]. However, CoGAPS uses a Markov chain Monte Carlo (MCMC) scheme for esti-

mating the BD model parameters, which is slower than the NMFs optimization algorithms implemented with the block-coordinate gradient descent scheme.

In order to address the lack of data mining functionalities and generality of current NMF toolboxes, we propose a general NMF toolbox in MATLAB which is implemented in two levels. The basic level is composed of the different variants of NMF, and the top level consists of the diverse data mining methods for biological data. The source code of this toolbox can be downloaded at `https://sites.google.com/site/nmftool` and `http://cs.uwindsor.ca/~li11112c/nmf`. The contributions of our toolbox are enumerated in the following:

1. The NMF algorithms are relatively complete and implemented in MATLAB. Since it is impossible and unnecessary to implement all NMF algorithms, we focus only on well-known NMF representatives. This repository of NMFs allows users to select the most suitable one in specific scenarios.

2. Our NMF toolbox includes many functionalities for mining biological data, such as clustering, biclustering, feature extraction, feature selection, and classification.

3. The toolbox also provides additional functions for biological data visualization, such as heat-maps and other visualization tools. They are pretty helpful for interpreting some results. Statistical methods are also included for comparing the performances of multiple methods.

2.2 Implementation

As mentioned above, this toolbox is implemented at two levels. The fundamental level is composed of several NMF variants and the advanced level includes many data mining approaches based on the fundamental level. The critical issues in implementing these NMF variants are addressed in this section. Table 3.1 summarizes all the NMF algorithms implemented in our toolbox. Users (researchers, students, and practitioners) should use the command `help nmfrule`, for example, in the command line, for help on how to select a given function and set its parameters.

Standard-NMF

The *standard-NMF* decomposes a non-negative matrix $\boldsymbol{X} \in \mathbb{R}^{m\times n}$ into two non-negative factors $\boldsymbol{A} \in \mathbb{R}^{m\times k}$ and $\boldsymbol{Y} \in \mathbb{R}^{k\times n}$ (where $k < \min\{m,n\}$), that is

$$\boldsymbol{X}_+ = \boldsymbol{A}_+\boldsymbol{Y}_+ + \boldsymbol{E}, \tag{3.1}$$

where, $\boldsymbol{E}$ is the error (or residual) and $\boldsymbol{M}_+$ indicates the matrix $\boldsymbol{M}$ is non-negative. Its optimization in the Euclidean space is formulated as

$$\min_{\boldsymbol{A},\boldsymbol{Y}} \frac{1}{2}\|\boldsymbol{X} - \boldsymbol{A}\boldsymbol{Y}\|_{\mathrm{F}}^2, \text{s.t. } ,\boldsymbol{A},\boldsymbol{Y} \geq 0. \tag{3.2}$$

Statistically speaking, this formulation is obtained from the log-likelihood function under the assumption of a Gaussian error. If multivariate data points are arranged in the columns of $\boldsymbol{X}$, then $\boldsymbol{A}$ is called the *basis*

Table 3.1: Algorithms of NMF variants.

Function	Description
nmfrule	The standard NMF optimized by gradient-descent-based multiplicative rules.
nmfnnls	The standard NMF optimized by NNLS active-set algorithm.
seminmfrule	Semi-NMF optimized by multiplicative rules.
seminmfnnls	Semi-NMF optimized by NNLS.
sparsenmfnnls	Sparse-NMF optimized by NNLS.
sparsenmfNNQP	Sparse-NMF optimized by NNQP.
sparseseminmfnnls	Sparse semi-NMF optimized by NNLS.
kernelnmfdecom	Kernel NMF through decomposing the kernel matrix of input data.
kernelseminmfrule	Kernel semi-NMF optimized by multiplicative rule.
kernelseminmfnnls	Kernel semi-NMF optimized by NNLS.
kernelsparseseminmfnnls	Kernel sparse semi-NMF optimized by NNLS.
kernelSparseNMFNNQP	Kernel sparse semi-NMF optimized by NNQP.
convexnmfrule	Convex-NMF optimized by multiplicative rules.
kernelconvexnmf	Kernel convex-NMF optimized by multiplicative rules.
orthnmfrule	Orth-NMF optimized by multiplicative rules.
wnmfrule	Weighted-NMF optimized by multiplicative rules.
sparsenmf2rule	Sparse-NMF on both factors optimized by multiplicative rules.
sparsenmf2nnqp	Sparse-NMF on both factors optimized by NNQP.
vsmf	Versatile sparse matrix factorization optimized by NNQP and l_1QP.
nmf	The omnibus of the above algorithms.
computeKernelMatrix	Compute the kernel matrix k(A,B) given a kernel function.

matrix and $\boldsymbol{Y}$ is called the *coefficient matrix*; each column of $\boldsymbol{A}$ is thus a *basis vector*. The interpretation is that each data point is a (sparse) non-negative linear combination of the basis vectors. It is well-known that the optimization objective is a non-convex optimization problem, and thus, *block-coordinate descent* is the main prescribed optimization technique for such problem. Multiplicative update rules were introduced in [37] for solving Equation (3.2). Though simple to implement, this algorithm is not guaranteed to converge to a stationary point [75]. Essentially the optimizations above, with respect to $\boldsymbol{A}$ and $\boldsymbol{Y}$, are *non-negative least squares* (NNLS). Therefore we implemented the alternating NNLS algorithm proposed in [75]. It can be proven that this algorithm converges to a stationary point. In our toolbox, functions `nmfrule` and `nmfnnls` are the implementations of the two algorithms above.

Semi-NMF

The standard NMF only works for non-negative data, which limits its applications. Ref. [32] extended it to s*emi-NMF* which removes the non-negative constraints on the data $\boldsymbol{X}$ and basis matrix $\boldsymbol{A}$. It can be expressed in the following equation:

$$\min_{\boldsymbol{A},\boldsymbol{Y}} \frac{1}{2}\|\boldsymbol{X}-\boldsymbol{A}\boldsymbol{Y}\|_{\mathrm{F}}^2, \text{ s.t. } \boldsymbol{Y} \geq 0. \tag{3.3}$$

Semi-NMF can be applied to the matrix of mixed signs, therefore it expands NMF to many fields. However, the gradient-descent-based update rule proposed in [32] is slow to converge (implemented in function

`seminmfrule` in our toolbox). Keeping $\boldsymbol{Y}$ fixed, updating $\boldsymbol{A}$ is a least squares problem which has an analytical solution

$$\boldsymbol{A} = \boldsymbol{X}\boldsymbol{Y}^{\mathrm{T}}(\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}})^{-1} = \boldsymbol{X}\boldsymbol{Y}^{\dagger}, \tag{3.4}$$

where $\boldsymbol{Y}^{\dagger} = \boldsymbol{Y}^{\mathrm{T}}(\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}})^{-1}$ is Moore-Penrose pseudoinverse. Updating $\boldsymbol{Y}$ while fixing $\boldsymbol{A}$ is a NNLS problem essentially as above. Therefore we implemented the fast NNLS based algorithm to optimize semi-NMF in function `seminmfnnls`.

Sparse-NMF

The standard NMF and semi-NMF have the issues of scale-variance and non-unique solutions, which imply that the non-negativity constrained on the least squares is insufficient in some cases. Sparsity is a popular regularization principle in statistical modeling [17], and has already been used in order to reduce the non-uniqueness of solutions and also and enhance interpretability of the NMF results. The *sparse-NMF* proposed in [34] is expressed in the following equation

$$\begin{aligned} &\min_{\boldsymbol{A},\boldsymbol{Y}} \frac{1}{2}\|\boldsymbol{X} - \boldsymbol{A}\boldsymbol{Y}\|_{\mathrm{F}}^2 + \frac{\eta}{2}\|\boldsymbol{A}\|_F^2 + \frac{\lambda}{2}\sum_{i=1}^{n}\|\boldsymbol{y}_i\|_1^2 \\ &\text{s.t. } \boldsymbol{A},\boldsymbol{Y} \geq 0, \end{aligned} \tag{3.5}$$

where, $\boldsymbol{y}_i$ is the i-th column of $\boldsymbol{Y}$. From the Bayesian perspective, this formulation is obtained from the log-posterior probability under the assumptions of Gaussian error, Gaussian-distributed basis vectors, and Laplace-distributed coefficient vectors. Keeping one matrix fixed and updating the other matrix can be formulated as a NNLS problem. In order to improve the interpretability of the basis vectors and speed up the algorithm, we implemented the following model instead:

$$\begin{aligned} &\min_{\boldsymbol{A},\boldsymbol{Y}} \frac{1}{2}\|\boldsymbol{X} - \boldsymbol{A}\boldsymbol{Y}\|_{\mathrm{F}}^2 + \lambda\sum_{i=1}^{n}\|\boldsymbol{y}_i\|_1 \\ &\text{s.t. } \boldsymbol{A},\boldsymbol{Y} \geq 0, \\ &\quad \|\boldsymbol{a}_i\|_2^2 = 1, \quad i = 1,\cdots,k. \end{aligned} \tag{3.6}$$

We optimize this using three alternating steps in each iteration. First, we optimize the following task:

$$\begin{aligned} &\min_{\boldsymbol{Y}} \frac{1}{2}\|\boldsymbol{X} - \boldsymbol{A}\boldsymbol{Y}\|_{\mathrm{F}}^2 + \lambda\sum_{i=1}^{n}\|\boldsymbol{y}_i\|_1 \\ &\text{s.t. } \boldsymbol{Y} \geq 0. \end{aligned} \tag{3.7}$$

then, $\boldsymbol{A}$ is updated as follows:

$$\begin{aligned} &\min_{\boldsymbol{A}} \frac{1}{2}\|\boldsymbol{X}-\boldsymbol{A}\boldsymbol{Y}\|_{\mathrm{F}}^2 \\ &\text{s.t. } \boldsymbol{A} \geq 0. \end{aligned} \tag{3.8}$$

and then, the columns of $\boldsymbol{A}$ are normalized to have unit l_2 norm. The first and second steps can be solved using *non-negative quadratic programming* (NNQP), whose general formulation is

$$\begin{aligned} &\min_{\boldsymbol{Z}} \sum_{i=1}^{n} \frac{1}{2}\boldsymbol{z}_i^{\mathrm{T}}\boldsymbol{H}\boldsymbol{z}_i + \boldsymbol{g}_i^{\mathrm{T}}\boldsymbol{z}_i + c_i \\ &\text{s.t. } \boldsymbol{Z} \geq 0, \end{aligned} \tag{3.9}$$

where, $\boldsymbol{z}_i$ is the i-th column of the variable matrix $\boldsymbol{Z}$. It is easy to prove that NNLS is a special case of NNQP. For example, Equation (3.7) can be rewritten as

$$\begin{aligned} &\min_{\boldsymbol{Y}} \sum_{i=1}^{n} \frac{1}{2}\boldsymbol{y}_i^{\mathrm{T}}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A})\boldsymbol{y}_i + (\lambda - \boldsymbol{A}^{\mathrm{T}}\boldsymbol{x}_i)^{\mathrm{T}}\boldsymbol{y}_i + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{x}_i \\ &\text{s.t. } \boldsymbol{Y} \geq 0. \end{aligned} \tag{3.10}$$

The implementations of the method in [34] and our method are given in functions `sparsenmfnnls` and `sparseNMFNNQP`, respectively. We also implemented the sparse semi-NMF in function `sparseseminmfnnls`.

Versatile Sparse Matrix Factorization

When the training data $\boldsymbol{X}$ is of mixed signs, the basis matrix $\boldsymbol{A}$ is not necessarily constrained to be non-negative; this depends on the application or the intentions of the users. However, without non-negativity, $\boldsymbol{A}$ is not sparse any more. In order to obtain sparse basis matrix $\boldsymbol{A}$ for some analysis, we may use l_1-norm on $\boldsymbol{A}$ to induce sparsity. The drawback of l_1-norm is that correlated variables may not be simultaneously non-zero in the l_1-induced sparse result. This is because l_1-norm is able to produce sparse but non-smooth results. It is known that l_2-norm is able to obtain smooth but non-sparse results. When both norms are used together, then correlated variables can be selected or removed simultaneously [33]. When smoothness is required on $\boldsymbol{Y}$, we may also use l_2-norm on it in some scenarios. We thus generalize the aforementioned NMF models into a

versatile form as expressed below

$$\begin{aligned}\min_{\boldsymbol{A},\boldsymbol{Y}} f(\boldsymbol{A},\boldsymbol{Y}) &= \frac{1}{2}\|\boldsymbol{X}-\boldsymbol{A}\boldsymbol{Y}\|_F^2 + \sum_{i=1}^{k}(\frac{\alpha_2}{2}\|\boldsymbol{a}_i\|_2^2 \\ &+\alpha_1\|\boldsymbol{a}_i\|_1) + \sum_{i=1}^{n}(\frac{\lambda_2}{2}\|\boldsymbol{y}_i\|_2^2+\lambda_1\|\boldsymbol{y}_i\|_1) \\ \text{s.t. } &\begin{cases}\boldsymbol{A}\geq 0 & \text{i.e., if } t_1=1\\ \boldsymbol{Y}\geq 0 & \text{i.e., if } t_2=1\end{cases},\end{aligned} \tag{3.11}$$

where, parameters: $\alpha_1 \geq 0$ controls the sparsity of the basis vectors; $\alpha_2 \geq 0$ controls the smoothness and the scale of the basis vectors; $\lambda_1 \geq 0$ controls the sparsity of the coefficient vectors; $\lambda_2 \geq 0$ controls the smoothness of the coefficient vectors; and, parameters t_1 and t_2 are boolean variables (0: false, 1: true) which indicate if non-negativity needs to be enforced on $\boldsymbol{A}$ or $\boldsymbol{Y}$, respectively. We can call this model *versatile sparse matrix factorization* (VSMF). It can be easily seen that the standard NMF, semi-NMF, and the sparse-NMFs are special cases of VSMF.

We devise the following multiplicative update rules for the VSMF model in the case of $t_1 = t_2 = 1$ (implemented in function `sparsenmf2rule`):

$$\begin{cases}\boldsymbol{A} = \boldsymbol{A} * \frac{\boldsymbol{X}\boldsymbol{Y}^{\mathrm{T}}}{\boldsymbol{A}\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}}+\alpha_2\boldsymbol{A}+\alpha_1} \\ \boldsymbol{Y} = \boldsymbol{Y} * \frac{\boldsymbol{A}^{\mathrm{T}}\boldsymbol{X}}{\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{Y}+\lambda_2\boldsymbol{Y}+\lambda_1}\end{cases}, \tag{3.12}$$

where, $\boldsymbol{A} * \boldsymbol{B}$ and $\frac{\boldsymbol{A}}{\boldsymbol{B}}$ are the element-wise multiplication and division operators of matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, respectively. Alternatively, we also devise an active-set algorithm for VSMF (implemented in function `vsmf`). When t_1(or t_2) $= 1$, $\boldsymbol{A}$ (or $\boldsymbol{Y}$) can be updated by NNQP (this case is also implemented in `sparsenmf2nnqp`). When t_1(or t_2) $= 0$, $\boldsymbol{A}$ (or $\boldsymbol{Y}$) can be updated using 1_1QP.

Kernel-NMF

Two features of a kernel approach are that i) it can represent complex patterns, and ii) the optimization of the model is dimension-free. We now show that NMF can also be kernelized.

The basis matrix is dependent on the dimension of the data, and it is difficult to represent it in a very high (even infinite) dimensional space. We notice that in the NNLS optimization, updating $\boldsymbol{Y}$ in Equation (3.10) needs only the inner products $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$, $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{X}$, and $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$. From Equation (3.4), we obtain $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} = (\boldsymbol{Y}^{\dagger})^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{Y}^{\dagger}$, $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{X} = (\boldsymbol{Y}^{\dagger})^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$. Therefore, we can see that only the inner product $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$ is needed in the optimization of NMF. Hence, we can obtain the kernel version, *kernel-NMF*, by replacing the inner product $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$ with a kernel matrix $K(\boldsymbol{X},\boldsymbol{X})$. Interested readers can refer to our recent paper [68] for further details. Based on the above derivations, we implemented the kernel semi-NMF using multiplicative update rule (in `kernelseminmfrule`) and NNLS (in `kernelseminmfnnls`). The sparse kernel semi-NMFs are implemented in functions `kernelsparseseminmfnnls` and `kernelSparseNMFNNQP` which are equivalent to each other.

The kernel method of decomposing a kernel matrix proposed in [76] is implemented in `kernelnmfdecom`.

Other Variants

Ref. [32] proposed the *Convex-NMF*, in which the columns of $\boldsymbol{A}$ are constrained to be the convex combinations of data points in $\boldsymbol{X}$. It is formulated as $\boldsymbol{X}_{\pm} = \boldsymbol{X}_{\pm}\boldsymbol{W}_{+}\boldsymbol{Y}_{+} + \boldsymbol{E}$, where $\boldsymbol{M}_{\pm}$ indicates that matrix $\boldsymbol{M}$ is of mixed signs. $\boldsymbol{XW} = \boldsymbol{A}$ and each column of $\boldsymbol{W}$ contains the convex coefficients of all the data points to get the corresponding column of $\boldsymbol{A}$. It has been demonstrated that the columns of $\boldsymbol{A}$ obtained with the convex-NMF are close to the real centroids of clusters. Convex-NMF can be kernelized as well [32]. We implemented the convex-NMF and its kernel version in `convexnmfrule` and `kernelconvexnmf`, respectively.

The basis vectors obtained with the above NMFs are non-orthogonal. Alternatively, *orthogonal NMF* (ortho-NMF) imposes the orthogonality constraint in order to enhance sparsity [77]. Its formulation is

$$\begin{aligned} &\boldsymbol{X} = \boldsymbol{ASY} + \boldsymbol{E} \\ &\text{s.t. } \boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} = \boldsymbol{I}, \quad \boldsymbol{YY}^{\mathrm{T}} = \boldsymbol{I}, \quad \boldsymbol{A}, \boldsymbol{S}, \boldsymbol{Y} \geq 0, \end{aligned} \tag{3.13}$$

where, the input $\boldsymbol{X}$ is non-negative, $\boldsymbol{S}$ absorbs the magnitude due to the normalization of $\boldsymbol{A}$ and $\boldsymbol{Y}$. Function `orthnmfrule` is its implementation in our toolbox. Ortho-NMF is very similar with the *non-negative sparse PCA* (NSPCA) proposed in [78]. The disjoint property on ortho-NMF may be too restrictive for many applications, therefore this property is relaxed in NSPCA. Ortho-NMF does not guarantee the maximum-variance property which is also relaxed in NSPCA. However NSPCA only enforces non-negativity on the basis vectors, even when the training data have negative values. We plan to devise a model in which the disjoint property, the maximum-variance property, the non-negativity and sparsity constraints can be controlled on both basis vectors and coefficient vectors.

There are two efficient ways of applying NMF on data containing missing values. First, the missing values can be estimated prior to running NMF. Alternatively, *weighted-NMF* [79] can be directly applied to decompose the data. Weighted-NMF puts a zero weight on the missing elements and hence only the non-missing data contributes to the final result. An expectation-maximization (EM) based missing value estimation during the execution of NMF may not be efficient. The weighted-NMF is given in our toolbox in function `wnmfrule`.

2.3 Results and Discussion

Based on the various implemented NMFs, a number of data mining tasks can be performed via our toolbox. Table 3.2 lists the data mining functionalities we provide in this level. These mining tasks are also described along with appropriate examples.

Clustering and Biclustering

NMF has been applied for clustering. Given data $\boldsymbol{X}$ with multivariate data points in the columns, the idea is that, after applying NMF on $\boldsymbol{X}$, a multivariate data point, say $\boldsymbol{x}_i$ is a non-negative linear combination of the

Table 3.2: NMF-based data mining approaches.

Function	Description
NMFCluster	Take the coefficient matrix produced by a NMF algorithm, and output the clustering result.
chooseBestk	Search the best number of clusters based on dispersion Coefficients.
biCluster	The biclustering method using one of the NMF algorithms.
featureExtractionTrain	General interface. Using training data, generate the bases of the NMF feature space.
featureExtractionTest	General interface. Map the test/unknown data into the feature space.
featureFilterNMF	On training data, select features by various NMFs.
featSel	Feature selection methods.
nnlsClassifier	The NNLS classifier.
perform	Evaluate the classifier performance.
changeClassLabels01	Change the class labels to be in $\{0,1,2,\cdots,C-1\}$ for C-class problem.
gridSearchUniverse	a framework to do line or grid search.
classificationTrain	Train a classifier, many classifiers are included.
classificationPredict	Predict the class labels of unknown samples via the model learned by classificationTrain.
multiClassifiers	Run multiple classifiers on the same training data.
cvExperiment	Conduct experiment of k-fold cross-validation on a data set.
significantAcc	Check if the given data size can obtain significant accuracy.
learnCurve	Fit the learning curve.
FriedmanTest	Friedman test with post-hoc Nemenyi test to compare multiple classifiers on multiple data sets.
plotNemenyiTest	Plot the CD diagram of Nemenyi test.
NMFHeatMap	Draw and save the heat maps of NMF clustering.
NMFBicHeatMap	Draw and save the heat maps of NMF biclustering.
plotBarError	Plot Bars with STD.
writeGeneList	write the gene list into a .txt file.
normmean0std1	Normalization to have mean 0 and STD 1.
sparsity	Calculate the sparsity of a matrix.
MAT2DAT	Write a data set from MATLAB into .dat format in order to be readable by other languages.

columns of $\boldsymbol{A}$; that is $\boldsymbol{x}_i \approx \boldsymbol{A}\boldsymbol{y}_i = y_{1i}\boldsymbol{a}_1 + \cdots + y_{ki}\boldsymbol{a}_k$. The largest coefficient in the i-th column of $\boldsymbol{Y}$ indicates the cluster this data point belongs to. The reason is that if the data points are mainly composed with the same basis vectors, they should therefore be in the same group. A basis vector is usually viewed as a cluster centroid or prototype. This approach has been used in [54] for clustering microarray data and in order to discover tumor subtypes. We implemented function `NMFCluster` through which various NMF algorithms can be selected. An example is provided in `exampleCluster` file in the folder of our toolbox.

The task of interpreting both the basis matrix and the coefficient is equivalent to simultaneously clustering the rows and columns of matrix $\boldsymbol{X}$. This is biclustering and the interested readers can refer to [80] for an excellent survey on biclustering algorithms and to [66] for a biclustering method based on NMF. We implemented a biclustering approach based on NMF in `biCluster` function. The bi-clusters can be visualized

via the function `NMFBicHeatMap`. We applied NMF to simultaneously grouping the genes and samples of a leukemia data set [54] which includes tumor samples of three subtypes. The goal is to find strongly correlated genes over a subset of samples. A subset of such genes and a subset of such samples form a bi-cluster. The heat-map is shown in Figure 3.1. Readers can find the script in `exampleBiCluster` file of our toolbox.

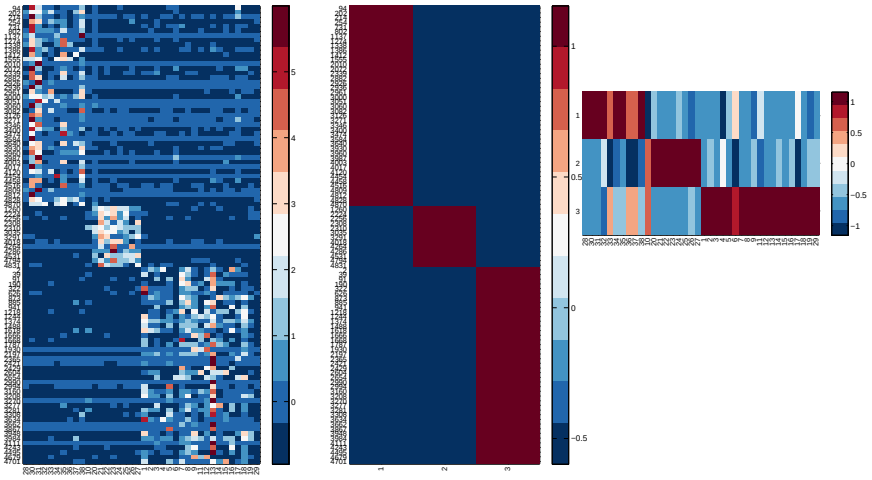


Figure 3.1: Heat map of NMF biclustering result. Left: the gene expression data where each column corresponds to a sample. Center: the basis matrix. Right: the coefficient matrix.

Basis Vector Analysis for Biological Process Discovery

We can obtain interesting and detailed interpretations via an appropriate analysis of the basis vectors. When applying NMF on a microarray data, the basis vectors are interpreted as potential biological processes [40] [34] [42]. In the following, we give one example for finding biological factors on gene-sample data, and two examples on time-series data. Please note they only serve as simple examples. Fine tuning of the parameters of NMF is necessary for accurate results.

First example: We ran our VSMF on the ALLAML gene-sample data of [54] with the settings $k = 3$, $\alpha_1 = 0.01$, $\alpha_2 = 0.01$, $\lambda_1 = 0$, $\lambda_2 = 0.01$, $t_1 = 1$, and $t_2 = 1$. Next, we obtain 81, 37, and 448 genes for the three factors, respectively. As in [34], we then performed gene set enrichment analysis (GSEA) by applying Onto-Express [81] on each of these sets of genes. Part of the result is shown in Table 3.3. We can see that the factor-specific genes selected by NMF correspond to some biological processes significantly. Please see file `exampleBioProcessGS` in the toolbox for details. GSEA can also be done using other tools, such as MIPS [82], GOTermFinder [83], and DAVID [84] [85].

Second example: We used NMF to cluster a time-series data of yeast metabolic cycle in [86]. Figure 3.2 shows the heat-map of NMF clustering, and Figure 3.3 shows the three basis vectors. We used `nmfnnls` function to decompose the data and `NMFHeatMap` to plot the heat-map. The detailed script is given in the

Table 3.3: Gene set enrichment analysis using Onto-Express for the factor specific genes identified by NMF.

Factor 1		Factor 2		Factor 3	
biological process	p-value	biological process	p-value	biological process	p-value
reproduction (5)	0	response to stimulus (15)	0.035	regulation of bio. proc. (226)	0.009
metabolic process (41)	0	biological regulation(14)	0.048	multi-organism proc. (39)	0.005
cellular process (58)	0			biological regulation (237)	0.026
death (5)	0				
developmental process (19)	0				
regulation of biological process (19)	0				

`exampleBioProcessTSYeast` file in the toolbox. We can clearly see that the three periodical biological processes corresponds exactly to the Ox (oxidative), R/B (reductive, building), and R/C (reductive, charging) processes discovered in [86].

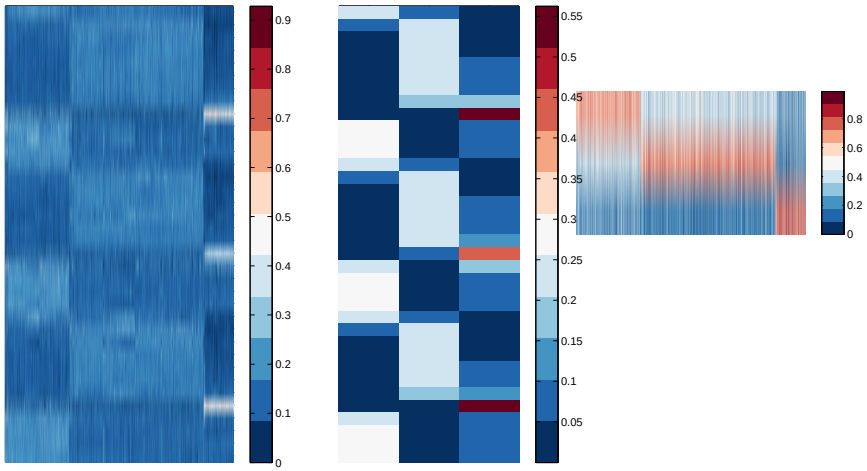


Figure 3.2: Heat map of NMF clustering result on yeast metabolic cycle time-series data. Left: the gene expression data where each column corresponds to a sample. Center: the basis matrix. Right: the coefficient matrix.

Third example: We used NMF to factorize a breast cancer time-series data set, which includes wild type MYCN cell lines and mutant MYCN cell lines [87]. The purpose of this example is to show that NMF is a potential tool to finding cancer drivers. One basic methodology is in the following. First, basis vectors are produced applying NMF on a time-series data. Then factor-specific genes are identified by computational or statistical methods. Finally, the regulators of these factor-specific genes are identified from any prior biological knowledge. This data set has 8 time points (0, 2, 4, 8, 12, 24, 36, 48 hr.). The zero time point is untreated and samples were collected at the subsequent time points after treatment with 4-hydroxytamoxifen (4-OHT). In our computational experiment, we use our VSMF implementation (function `vsmf`). we set $k = 2$. Because this data set has negative values we set $t_1 = 0$ and $t_2 = 1$. We set $\alpha_1 = 0.01$, $\alpha_2 = 0$, $\lambda_1 = 0$, and $\lambda_2 = 0.01$. The basis vectors of both wild-type and mutant data are compared in Figure 3.4. From the wild-

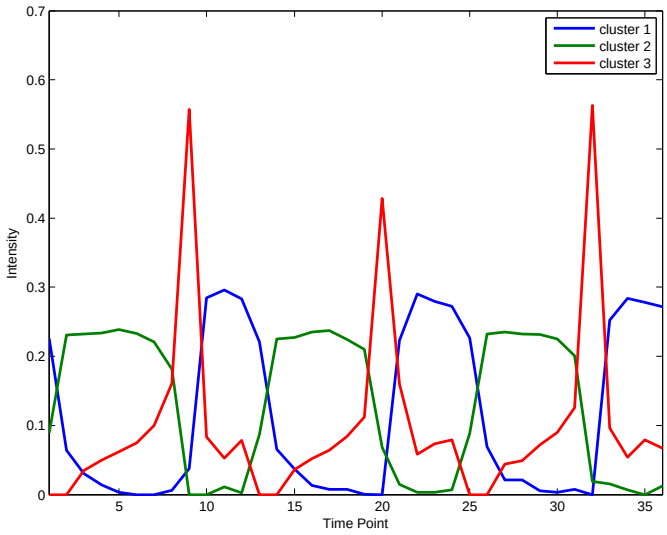


Figure 3.3: Biological processes discovered by NMF on yeast metabolic cycle time-series data.

type time-series data, we can successfully identify two patterns. The rising pattern corresponds to the induced signature and the falling pattern corresponding to the repressed signature in [87]. It is reported in [87] that the MYC target genes contributes to both patterns. From the mutant time-series, we can obtain two flat processes, which are reasonable. The source code of this example can be found in `exampleBioProcessMYC`. We also recommend the readers to see the methods based on matrix decompositions which are proposed in [42] and [41] and devised for identifying signaling pathways.

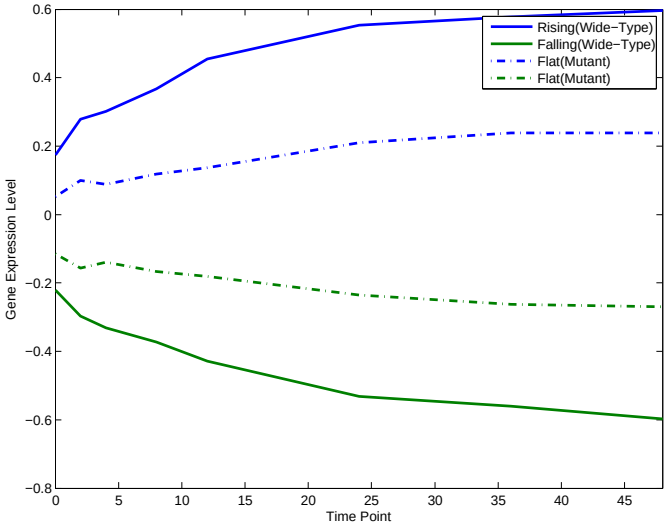


Figure 3.4: Biological processes discovered by NMF on breast cancer time-series data.

Basis Vector Analysis for Gene Selection

The columns of $\boldsymbol{A}$ for a gene expression data set are called *metasamples* in [54]. They can be interpreted as biological processes, because their values imply the activation or inhibition of some the genes. Gene selection aims to find marker genes for disease prediction and to understand the pathways they contribute to. Rather than selecting genes on the original data, the novel idea is to conduct gene selection on the metasamples. The reason is that the discovered biological processes via NMF are biologically meaningful for class discrimination in disease prediction, and the genes expressed differentially across these processes contribute to better classification performance in terms of accuracy. In Figure 3.1 for example, three biological processes are discovered and only the selected genes are shown. We have implemented the information-entropy-based gene selection approach proposed in [34] in function `featureFilterNMF`. We give an example on how to call this function in file `exampleFeatureSelection`. It has been reported that it can select meaningful genes, which has been verified with gene ontology analysis. Feature selection based on supervised NMF will also be implemented.

Feature Extraction

Microarray data and mass spectrometry data have tens of thousands of features but only tens or hundreds of samples. This leads to the issues of *curse of dimensionality*. For example, it is impossible to estimate the parameters of some statistical models since the number of their parameters grow exponentially as the dimension increases. Another issue is that biological data are usually noisy; which crucially affects the performances of classifiers applied on the data. In cancer study, a common hypothesis is that only a few biological factors (such as the oncogenes) play a crucial role in the development of a given cancer. When we generate data from control and sick patients, the high-dimensional data will contain a large number of irrelevant or redundant information. Orthogonal factors obtained with *principal component analysis* (PCA) or *independent component analysis* (ICA) are not appropriate in most cases. Since NMF generates non-orthogonal (and non-negative) factors, therefore it is much reasonable to extract important and interesting features from such data using NMF. As mentioned above, training data $\boldsymbol{X}_{m\times n}$, with m features and n samples, can be decomposed into k metasamples $\boldsymbol{A}_{m\times k}$ and $\boldsymbol{Y}_{k\times n}$, that is

$$\boldsymbol{X} \approx \boldsymbol{A}\boldsymbol{Y}_{\text{trace}},\ \text{s.t. } \boldsymbol{A}, \boldsymbol{Y}_{\text{trace}} \geq 0, \tag{3.14}$$

where, $\boldsymbol{Y}_{\text{trace}}$ means that $\boldsymbol{Y}$ is obtained from the training data. The k columns of $\boldsymbol{A}$ span the k-dimensional *feature space* and each column of $\boldsymbol{Y}_{\text{trace}}$ is the representation of the corresponding original training sample in the feature space. In order to project the p unknown samples $\boldsymbol{S}_{m\times p}$ into this feature space, we have to solve the following non-negative least squares problem:

$$\boldsymbol{S} \approx \boldsymbol{A}\boldsymbol{Y}_{\text{uk}},\ \text{s.t. } \boldsymbol{Y}_{\text{uk}} \geq 0, \tag{3.15}$$

where, $\boldsymbol{Y}_{\text{uk}}$ means the $\boldsymbol{Y}$ is obtained from the unknown samples. After obtaining $\boldsymbol{Y}_{\text{trace}}$ and $\boldsymbol{Y}_{\text{uk}}$, the learning and prediction steps can be done quickly in the k-dimensional feature space instead of the m-dimensional

original space. A classifier can learn over $\boldsymbol{Y}_{\text{trace}}$, and then predicts the class labels of the representations of unknown samples, that is $\boldsymbol{Y}_{\text{uk}}$.

From the aspect of interpretation, the advantage of NMF over PCA and ICA is that the metasamples are very useful in the understanding of the underlying biological processes, as mentioned above.

We have implemented a pair of functions `featureExtractionTrain` and `featureExtractionTest` including many linear and kernel NMF algorithms. The basis matrix (or, the inner product of basis matrices in the kernel case) is learned from the training data via the function `featureExtractionTrain`, and the unknown samples can be projected onto the feature space via the function `featureExtractionTest`. We give examples of how to use these functions in files `exampleFeatureExtraction` and `exampleFeatureExtractionKernel`.

Figure 3.5 shows the classification performance of SVM *without* dimension reduction and SVM *with* dimension reduction using linear NMF, kernel NMF with *radial basis function* (RBF) kernel, and PCA on two data sets, SRBCT [59] and Breast [29]. Since ICA is computationally costly, we did not include it in the comparisons. The bars represent the averaged 4-fold cross-validation accuracies using *support vector machine* (SVM) as classifier over 20 runs. We can see that NMF is comparable to PCA on SRBCT, and is slightly better than PCA on Breast data. Also, with only few factors, the performance after dimension reduction using NMF is at least comparable to that without using any dimension reduction. As future work, supervised NMF will be investigated and implemented in order to extract discriminative features.

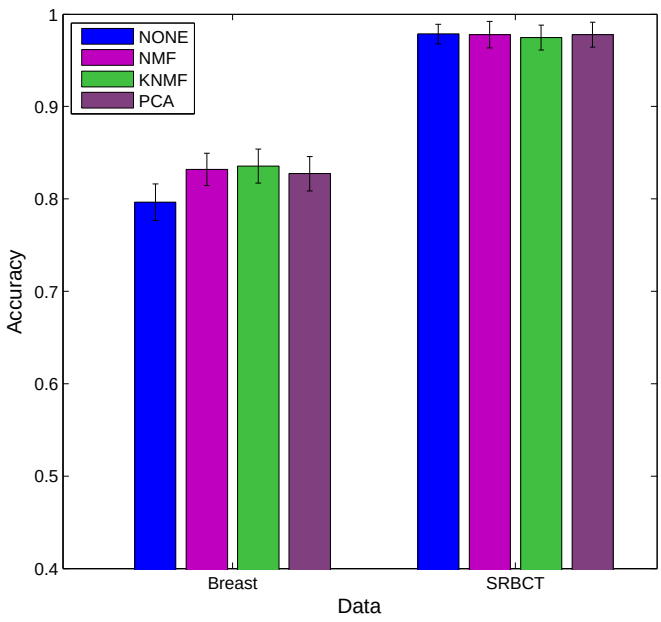


Figure 3.5: Mean accuracy and standard deviation results of NMF-based feature extraction on SRBCT data

Classification

If we make the assumption that every unknown sample is a sparse non-negative linear combination of the training samples, then we can directly derive a classifier from NMF. Indeed, this is a specific case of NMF in which the training samples are the basis vectors. Since the optimization process within NMF is a NNLS problem, we call this classification approach the *NNLS classifier* [88]. A NNLS problem is essentially a quadratic programming problem as formulated in Equation (3.9), therefore, only inner products are needed for the optimization. We thus can naturally extend the NNLS classifier to kernel version. Two features of this approach are that: i) the sparsity regularization help avoid overfitting the model; and ii) the kernelization allows a dimension-free optimization and also linearizes the non-linear complex patterns within the data. The implementation of the NNLS classifier is in file `nnlsClassifier`. Our toolbox also provides many other classification approaches including SVM classifier. Please see file `exampleClassification` for demonstration. In our experiment of 4-fold cross-validation, accuracies of 0.7865 and 0.7804 are respectively obtained with linear and kernel (RBF) NNLS classifier on Breast data set. They achieved accuracies of 0.9762 and 0.9785, respectively, over SRBCT data.

Biological data are usually noisy and sometimes contains missing values. A strength of the NNLS classifier are that it is robust to noise and to missing values, making NNLS classifiers quite suitable for classifying biological data [88].

In order to show its robustness to noise, we added a Gaussian noise of mean 0 and variance from 0 to 4 with increment 0.5 on SRBCT. Figure 3.6 illustrates the results of NNLS, SVM, and *1-nearest neighbor* (1-NN) classifiers using this noisy data. It can be seen that as the noise increases, NNLS outperforms SVM and 1-NN significantly.

To deal with the missing value problem, three strategies are usually used: incomplete sample or feature removal, missing value imputation (i.e., estimation), and ignoring missing values. Removal methods may delete important or useful information for classification and particularly when there is a large percentage of missing values in the data. Imputation methods may create false data depending on the magnitude of the true estimation errors. The third method ignores using the missing values during classification. Our approach in dealing with the missing value problem is also to ignore them. The NNLS optimization needs only the inner products of pairs of samples. Thus, when computing the inner product of two samples, say $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, we normalize them to have unit l_2-norm using only the features present in both samples, and then we take their inner product. As an example, we randomly removed between 10% to 70% of data values in STBCT data. Using such incomplete data, we compared our method with the zero-imputation method (that is, estimating all missing values as 0). In Figure 3.7, we can see that the NNLS classifier using our missing value approach outperforms the zero-imputation method in the case of large missing rate. Also, the more sophisticated k-nearest neighbor imputation (KNNimpute) method [89] will fail on data with in high percentage of missing values.

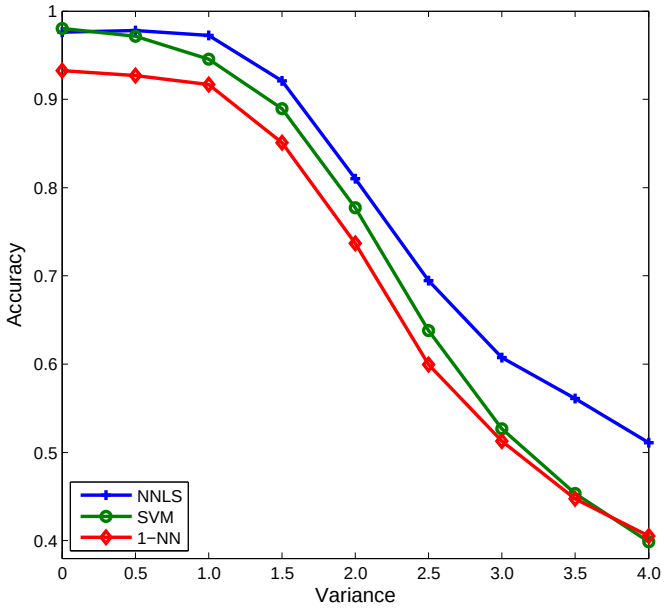


Figure 3.6: The mean accuracy results of NNLS classifier for different amount of noise on SRBCT data.

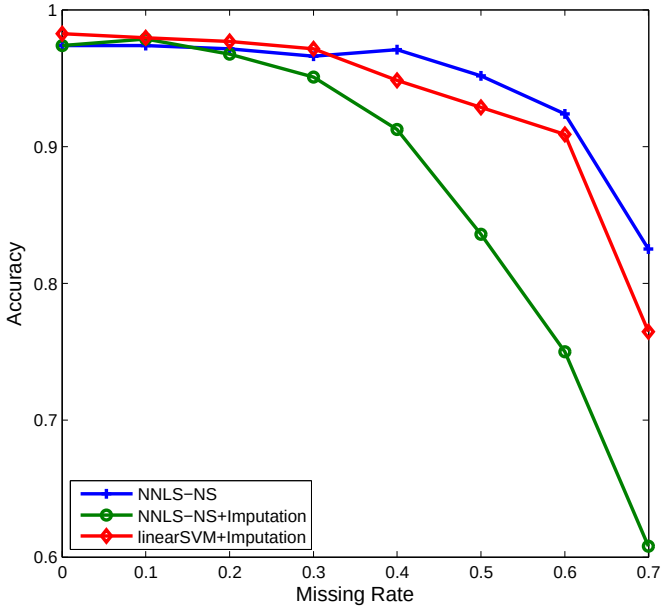


Figure 3.7: The mean accuracy results of NNLS classifier for different missing value rates on SRBCT data.

Statistical Comparison

The toolbox provides two methods for statistical comparisons and evaluations of different methods. The first is a two-stage method proposed in [45]. The importance of this method is that it can estimate the data-size

requirement for attaining a significant accuracy and extrapolate the performance based on the current available data. Generating biological data is usually very expensive and thus this method can help researchers to evaluate the necessity of producing more data. At the first stage, the minimum data size required for obtaining a significant accuracy is estimated. This is implemented in function `significantAcc`. The second stage is to fit the learning curve using the error rates of large data sizes. It is implemented in function `learnCurve`. In our experiments, we found that the NNLS classifier usually requires fewer number of samples for obtaining a significant accuracy. For example on SRBCT data, NNLS requires only 4 training samples while SVM needs 19 training samples. The fitted learning curves of NNLS and SVM classifiers are shown in Figure 3.8. We provide an example of how to plot this figure in file `exampleFitLearnCurve`.

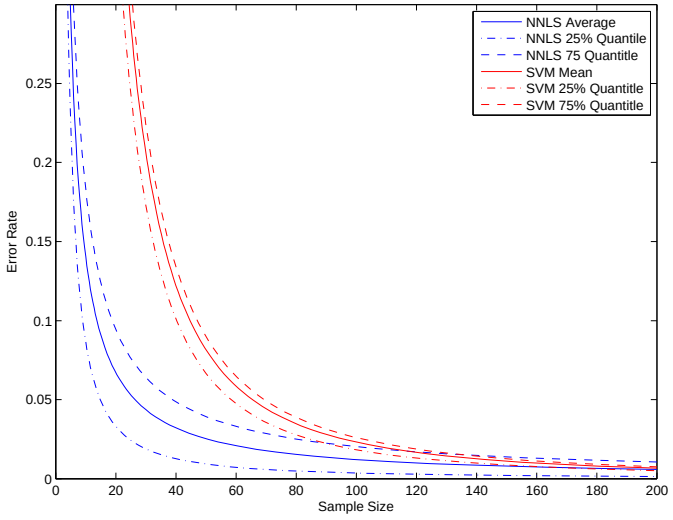


Figure 3.8: The fitted learning curves of NNLS and SVM classifiers on SRBCT data.

The second method is the nonparametric Friedman test coupled with post-hoc Nemenyi test to compare multiple classifiers over multiple data sets [64]. It is difficult to draw an overall conclusion if we compare multiple approaches in a pairwise fashion. Friedman test has been recommended in [64] because it is simple, safe and robust, compared with parametric tests. It is implemented in function `FriedmanTest`. The result can be presented graphically using the *crucial difference* (CD) diagram as implemented in function `plotNemenyiTest`. CD is determined by significant level α. Figure 3.9 is an example of the result of the Nemenyi test for comparing 8 classifiers over 13 high dimensional biological data sets. This example can be found in file `exampleFriedmanTest`. If the distance of two methods is greater than the CD then we conclude that they differ significantly.

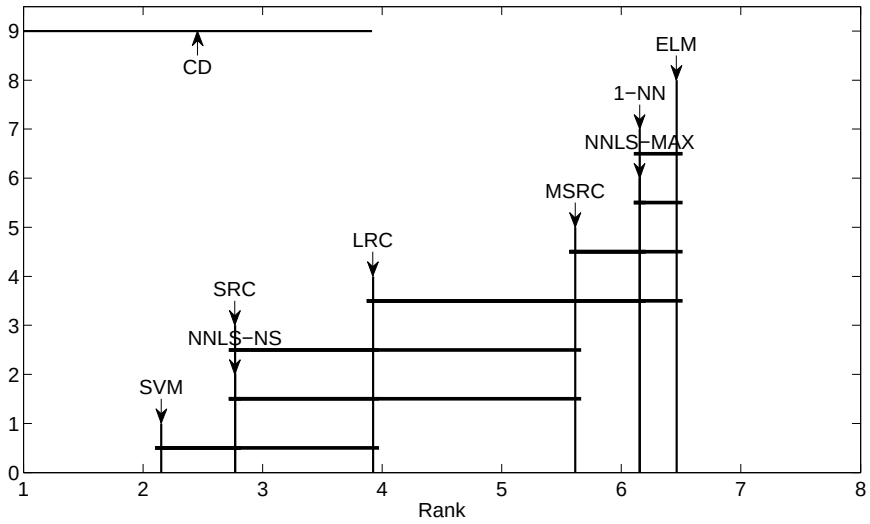


Figure 3.9: Nemenyi test comparing 8 classifiers over 13 high dimensional biological data ($\boldsymbol{\alpha = 0.05}$).

3 The Sparse Representation Toolbox

3.1 Introduction

We have implemented all the sparse representation methods mentioned in the previous section. We pack these implementations all together in our *Sparse Representation Toolbox* (SR toolbox). We separate our code into basis level and advanced level. The basis level is mainly composed of sparse coding models, dictionary learning models, and the corresponding optimization. The advanced level includes kernel classification and feature extraction methods based on sparse representation. We introduce both levels in the subsequent sections. Examples of using these functions can be found in the folder of the source code. The source code of this toolbox can be downloaded at `https://sites.google.com/site/sparseReptool` and `http://cs.uwindsor.ca/~li11112c/sr`.

3.2 Implementations

The Basis Implementation

We summarize our implementations of the basic sparse representation techniques in Table 3.4. Functions `KSRSC` and `KSRDL` are the generic interfaces of kernel sparse coding and dictionary learning. In files `exampleKSRSC`, and `exampleKSRDL`, we provide examples of how to uses these functions, respectively. Function `vsmf` is a more general, as has been introduced in the NMF toolbox. In Table 3.4, we also list our implementations of active-set, interior-point, proximal, and decomposition methods for sparse coding. In file `exampleOptSC`, examples are given to show usages of these optimization functions.

Table 3.4: Methods of sparse representation.

Function	Description
KSRSC	Kernel sparse coding methods including l_1LS, NNLS, and l_1NNLS.
KSRDL	The generic kernel dictionary learning framework.
vsmf	Versatile sparse matrix factorization optimized by NNQP and l_1QP.
l1QPActiveSet	The active-set method for single or multiple l_1QP problem.
NNQPActiveSet	The active-set method for single or multiple NNQP problem.
l1QPIP	The interior-point method for single l_1QP problem.
l1QPIPMulti	The interior-point method for multiple l_1QP problem.
NNQPIP	The interior-point method for single NNQP problem.
NNQPIPMulti	The interior-point method for multiple NNQP problem.
l1QPProximal	The proximal method for single l_1QP problem.
l1QPProximalMulti	The proximal method for l_1QP problem.
l1QPSMO	The SMO method for single l_1QP problem.
l1QPSMOMulti	The SMO method for multiple l_1QP problem.
NNQPSMO	The SMO method for single NNQP problem.
NNQPSMOMulti	The SMO method for multiple NNQP problem.
computeKernelMatrix	Compute the kernel matrix k(A,B) given a kernel function.
sparsity	Computer the sparsity of a matrix.
normalizeKernelMatrix	Normalize kernel matrices to let each sample have unit l_2-norm.

The Advanced Implementation

The advanced level of our implementation consists of machine learning applications based on sparse representation. The breakdown of these applications is given in Table 3.5. First of all, using function `KSRSCClassifier`, one can conduct kernel-sparse-coding based classification. The l_1LS, NNLS, l_1LS models with different optimization algorithms and kernels can be specified by the input of `KSRSCClassifier`. Function `computeMetaSample` implements the training of MNNLS, that is learning the sub-dictionaries. The prediction of MNNLS can be conducted by `KSRSCClassifier`. Function `lrc` is the implementation of kernel *linear regression classification* (LRC) approach which extends the original LRC method [90]. Functions `nearestCentroidTrain` and `nearestCentroidPredict` implement the training and prediction phases of the kernel *nearest centroid* (NC) method. Functions `classificationTrain` and `classificationPredict` are the unified interfaces of these classifiers. Users can easily add their own classifiers in these interfaces. Function `multiClassifiers` allows users to choose multiple classifiers. Cross-validation can be conducted by function `cvExperiment`. The examples of all these methods are given in file `exampleClassification`.

Second, function `featureExtractionTrain` is the unified interface of the training of feature extraction based on dictionary learning. Function `featureExtractionTest` is used after `featureExtractionTrain` to project the unknown samples to the feature space learned by `featureExtractionTrain`. We provide an example of feature extraction in file `exampleFeatExtr`.

Table 3.5: Sparse-representation based machine learning methods.

Function	Description
KSRSCClassifier	The classification approach based on kernel sparse coding.
computeMetaSample	The training of sub-dictionary learning.
lrc	Linear regression classification.
nearestCentroidTrain	Train the kernel nearest centroid classifier.
nearestCentroidPredict	Predict the class labels of unknown samples by the model trained by nearestCentroidTrain.
classificationTrain	Train a classifier, many classifiers are included.
classificationPredict	Predict the class labels of unknown samples via the model learned by classificationTrain.
multiClassifiers	Run multiple classifiers on the same training data.
cvExperiment	Conduct experiment of k-fold cross-validation on a data set.
featureExtractionTrain	General interface. Using training data, generate the bases of the SR feature space.
featureExtractionTest	General interface. Map the test/unknown data into the feature space.
subspace	The nearest subspace rule used in KSRSCClassifier.
knnrule	The weighted k-nearest neighbor rule used in KSRSCClassifier.
leaveMOut	Leave m out.
changeClassLabels01	Change the class labels to be in $\{0,1,2,\cdots,C-1\}$ for C-class problem.
perform	Compute the performance of classification.

3.3 Conclusions

We have developed the NMF toolbox and SR toolbox to accommodated our implementations of sparse representation techniques. The basic levels allow the machine learning researchers to devise new methods based on the basic techniques. Researchers from the other areas, such as signal processing and computer vision, can also conveniently apply them. The advanced levels facilitate the bioinformaticians to conduct analysis including clustering, feature extraction, feature selection, and classification. Examples are provided to demonstrate the usages of these functionalities.

We mention some of our future works below. First, we will include more NMF algorithms such as nsNMF, LS-NMF, and supervised NMF. Second, we are very interested in implementing and speeding up the Bayesian decomposition method which is actually a probabilistic NMF introduced independently in the same period as the standard NMF. Third, we would like to implement Markov chain Monte Carlo method for the optimization of NMF and sparse representation. Finally, we plan to realize a supervised dictionary learning based on Bayesian regression.