ENSTA ParisTech - Université Paris Sud

Final year internship

Operational Feature Selection in Gaussian Mixture Models

Adrien Lagrange

supervised by
Mathieu Fauvel (Dynafor, ENSAT/INRA)
Manuel Grizonnet (CNES)

August 25, 2016

Abstract

This report presents a forward feature selection algorithm based on Gaussian mixture model (GMM) classifiers. The algorithm selects iteratively features that maximizes a criterion function which can be either a classification rate or a measure of divergence. We explore several variations of this algorithm by changing the criterion function and also by testing a floating forward variation allowing backward step to discard already selected features.

An important effort is made in exploiting GMM properties to implement a fast algorithm. In particular, update rules of the GMM model are used to compute the criterion function with various sets of features. The result is a C++ remote module for the remote sensing processing toolbox Orfeo (OTB) developed by CNES.

Finally, the method is tested and also compared to other classifiers using two different datasets, one of hyperspectral images with a lot of spectral variables and one with heterogeneous spatial features. The results validate the fact that the method performs well in terms of processing time and classification accuracy in comparison to the standard classifiers available in OTB.

Keywords: remote sensing, hyperspectral imaging, feature selection, gaussian mixture model, fast computing.

Contents

List of Figures

1. Introduction

1.1. Context

With the increasing number of remote sensing missions, the quantity of data available for a given landscape becomes larger and larger. Several missions are at the verge of producing huge amount of data or already produced it. After 2018, the EnMAP (Environmental Mapping and Analysis Program) satellites missioned by the German space agency will produced images with 244 spectral bands with a resolution of 30x30m and revisit every 4 days[1]. The Hyperspectral Infrared Imager (HyspIRI) of NASA will also take images with 212 spectral bands every 5 days. Additionally to hyperspectral data, hypertemporal data are also developping. The European satellites Sentinel-2 were launched successfully recently and the hypertemporal data produced by this mission will be fully available at the end of 2016[2]. A quick presentation of hyperspectral images is avaible in Appendix A.

Therefore, data is more and more difficult to process because of statistical and computational issues. We often refer to this statistical issues as the *curse of dimansionality*. The main problem is the fast increase of the number of parameters to estimate in order to build a model when dimension expands [1]. Thus, a important number of labeled samples is needed to perform learning. For example, in the case of Gaussian Mixture Models, there are $\frac{d(d+3)}{2}$ parameters for each class if d stands for the dimension of the samples. It means that, with 200 describing variables, the model estimation requires at least 20,300 samples by class.

The computational issues are multiple. The computing infrastructure needed to process data is more and more expensive because the processing might requires a GPU and a large amount of RAM to load images which can weight a dozen of gigabytes [2][3]. The processing time is also limiting and requires to use parallelize computing (GPU, multi-threading). Hence, in many remote sensing applications, the extraction of features from a large amount of available data is required [4].

For instance, in land-cover classification, given a set of spatial, temporal and spectral features, it is possible to extract those which are the most discriminant for the purpose of classification [5]. In hyperspectral data analysis from the hundreds of available spectral channels, it is possible to reduce the number of channels to make the processing more efficient in terms of statistical complexity because of the reduction of the number of parameters to estimate and thus also in term of computational time. Moreover, dimensional reduction might improves the capacity of generalization of the classifier and avoid overfitting.

There are two ways to reduce dimension [6]: features extraction and feature selection. Feature extraction means creating new features by combining the existing ones, for example linear combination as in Principal Component Analysis [4]. To the contrary, features selection selects a subset of existing features. It has the advantage to be much more interpretable for the end-user. The selected subset of features corresponds to the most important features for the given task.

There is a large diversity of methods for feature selection. However, they usually do not scale well with the number of pixels to be processed [7]. Nevertheless, methods based on Gaussian Mixture Models (GMM) have several interesting properties that make them suitable for feature selection in the context of large amount of data. By taking advantage of their intrinsic properties, it is possible to increase the computational efficiency with respect to standard implementation.

This work proposes to develop a forward feature extraction method using GMM in continuation of [7] and an upgraded floating forward method. The first method selects iteratively the meaningful features and the second introduces possible backward steps after each addition of a feature. An efficient implementation of the method is presented in order to handle large amount of data. Moreover, the developed algorithm is made available as a remote module of the C++ Orfeo Toolbox [8].

[1] http://www.enmap.org/
[2] https://sentinel2.cnes.fr/en/sentinel-2-0

Finally, tests are conducted to compare different variations of the method and to compare also to other classifiers (GMM, Random Forest, k-nearest-neighbor) available in the Orfeo Toolbox.

The remaining of this report is organized as follows. The remaining of Section 1 explains the organization and objectives of the internship. Then Section 2 presents Gaussian Mixture Model classifiers and two methods to make them suitable for high-dimension space. First method which is regularization of the model is introduced in Section 2.2. Second method is features selection and in Section 2.3, the feature selection method at the core of our work is presented. The work done to develop a smart implementation of our selection method is presented in Section 3. And finally, the tests conducted to explore the various of our algorithm and to compare it to other standard classifier are detailed in Section 4.

1.2. Objectives

Several outcomes are expected of this internship. First, based on the previous work [7] of M.Fauvel, we want to **reproduce the sequential forward features selection method and develop an upgrade version** corresponding to a floating forward selection allowing backward step during selection. This first part will results in a Python code freely available on Github [3].

Then, the aim is to collaborate with CNES in order to **produce an external module which could be plugged in the open-source library Orfeo Toolbox (OTB) developed by CNES** ([8]). This module implements the same methods as the Python version but has to be written in C++ and be compatible with the OTB. The module obtained is a fork of a template furnished by OTB developers and is also available freely on Github [4].

Finally, we want to **test the efficiency of this selection method and compare it to classical classifiers**. The testing is conducted first on hyperspectral images where features are directly the spectral band and secondly on remote sensing images where numerous and various features are used (texture features, morphological profiles, radiometric indexes, ...).

1.3. Planning

As can be expected, the cycle of development is conducted from high-level to low-level which means that the Python code is first developed and so allow us to set definitively the structure of the algorithm. The low-level C++ code is then developed in order to assure good performances and to respect the interface with the OTB.

Each implementation is followed by a testing period during which the code is tested with artificial and easily manipulable data. Moreover, the development of the C++ code is done in interaction with developers of the OTB and, in particular, a first meeting is organized at the early stage of the development to assure that the best choice are made and a second meeting is set during the validation of the C++ code to get feedbacks.

The experimentation are conducted at the end to demonstrate the interest of the project. The Gantt diagram 1 summarized the organization of the project.

1.4. Internship context

The UMR DYNAFOR, created in 2003, depends of the National Institute of Agricultural Research (INRA) and National Polytechnic Institute of Toulouse (INPT). The lab is specialized in landscape ecology and is composed of 36 permanents employees and 9 PhD students. The lab objective is to assure the sustainable management of forest resources, the biodiversity conservation and to study ecosystem services in rural areas. To reach its goals, the lab is divided in three axes:

[3] https://github.com/Laadr/FFFS

[4] https://github.com/Laadr/otbExternalFastFeaturesSelection

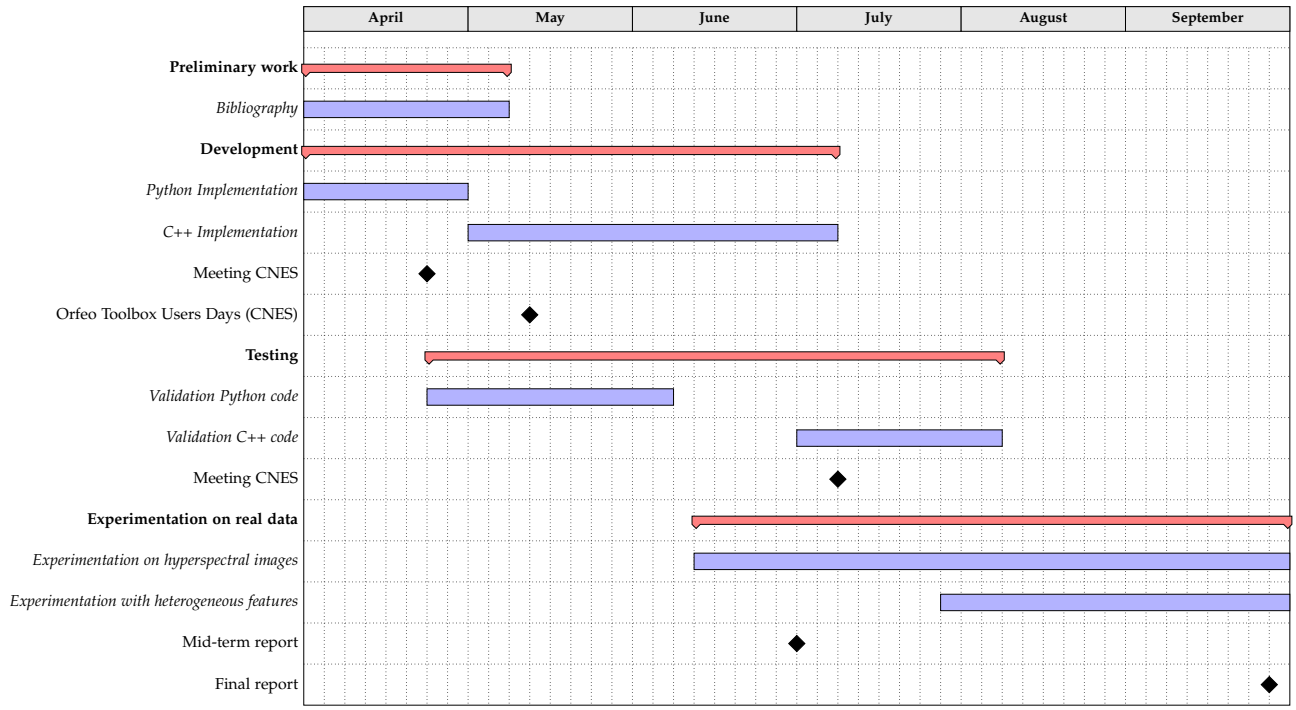


Figure 1: Gantt diagram of the internship.

- RAMSSES (in French: Recherche en Analyse et Modélisation de Systèmes Socio-Ecologiques Spatialisés) focus on remote sensing, geomatics and modelling,
- BIOFOR (Biodiversité des forêts rurales et des milieux semi-naturels dans les paysages) focus on biodiversity, forest areas and fauna,
- SECOTEAM (Services écosystémiques de la biodiversité dans les paysages agricoles) focus on ecosystem services and interactions between agents.

The following work is part of the mission of the RAMSSES axis and aims to develop a tool useful for the processing of hyperspectral images which supports the works done by the other axes.

2. Gaussian Mixture Models in high dimension space

In the remaining, the following notations are used. $\mathcal{S} = \{\mathbf{x}_i, y_i\}_{i=1}^n$ is the training set where $\mathbf{x}_i \in \mathbb{R}^d$ is the vector of features of the i^{th} sample, $y_i = 1, ..., C$ the associated label, C the number of labels, n the number of samples and n_c the number of samples of class c.

2.1. Gaussian Mixture Models

The hypothesis of mixture models is that a given sample is the realization of a random vector which distribution is a mixture (convex combination) of several class conditioned distribution:

$$p(\mathbf{x}) = \sum_{c=1}^{C} \pi_c f_c(\mathbf{x}|\theta) \tag{1}$$

where π_c is the prior i.e. the proportion of class c and f_c a probability density function parametrized by θ.

The Gaussian mixture model (GMM) assumes that each f_c is , conditionally to c, a Gaussian distribution of parameters $\boldsymbol{\mu}_c$ and $\boldsymbol{\Sigma}_c$ and so $f_c(\mathbf{x}|\theta)$ can be written

$$f_c(\mathbf{x}|\theta) = \frac{1}{(2\pi)^{\frac{d}{2}}|\boldsymbol{\Sigma}_c|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_c)^t\boldsymbol{\Sigma}_c^{-1}(\mathbf{x}-\boldsymbol{\mu}_c)\right).$$

Such a model is used in the case of supervised learning and the class parameters $\boldsymbol{\mu}_c$ and $\boldsymbol{\Sigma}_c$ can be estimated using the training samples. In our work, we choose to compute them with the conventional unbiased empirical estimator

$$\hat{\pi}_c = \frac{n_c}{n}, \tag{2}$$

$$\hat{\boldsymbol{\mu}}_c = \frac{1}{n_c}\sum_{\{i|y_i=c\}} \mathbf{x}_i, \tag{3}$$

$$\hat{\boldsymbol{\Sigma}}_c = \frac{1}{(n_c-1)}\sum_{\{i|y_i=c\}} (\mathbf{x}_i-\boldsymbol{\mu}_c)(x_i-\boldsymbol{\mu}_c)^t. \tag{4}$$

To determine the class of a samples, the maximum a posteriori (MAP) rule is used and thus, if we simplify with Bayes' law, the decision rule can be written as

$$\begin{aligned}\mathbf{x} \text{ belongs to } c &\Leftrightarrow c = \arg\max_{c\in C} p(c|\mathbf{x}),\\ &\Leftrightarrow c = \arg\max_{c\in C} \frac{p(c)p(\mathbf{x}|c)}{p(\mathbf{x})},\\ &\Leftrightarrow c = \arg\max_{c\in C} p(c)p(\mathbf{x}|c).\end{aligned}$$

By taking the log, a simplified decision formula is obtained

$$\begin{aligned}Q_c(\mathbf{x}) &= 2\log\left(p(c)p(\mathbf{x}|c)\right)\\ &= 2\log\left(\pi_c\frac{1}{(2\pi)^{\frac{d}{2}}|\boldsymbol{\Sigma}_c|^{\frac{1}{2}}}\exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_c)^t\boldsymbol{\Sigma}_c^{-1}(\mathbf{x}-\boldsymbol{\mu}_c)\right)\right)\\ &= 2\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_c)^t\boldsymbol{\Sigma}_c^{-1}(\mathbf{x}-\boldsymbol{\mu}_c)\right) + 2\log\left(\frac{1}{(2\pi)^{\frac{d}{2}}|\boldsymbol{\Sigma}_c|^{\frac{1}{2}}}\right) + 2\log(\pi_c)\\ &= -(\mathbf{x}-\boldsymbol{\mu}_c)^t\boldsymbol{\Sigma}_c^{-1}(\mathbf{x}-\boldsymbol{\mu}_c) - \log(|\boldsymbol{\Sigma}_c|) + 2\log(\pi_c) - d\log(2\pi).\end{aligned} \tag{5}$$

It is interesting to notice that the covariance, its inverse and its determinant are a key element of the decision function and that the estimation of these elements suffer from the curse of dimensionality. More precisely, the number of parameters to estimate (mean vectors, covariance matrices, proportions) increases quadratically relatively to the number of features and we need at least as many samples as parameters to make an estimation which can be an issue. For example, with hyperspectral data, we face high dimensional samples but very few labeled samples are available because of the difficulty and the cost to collect ground-truth.

A lack of samples induces badly-conditioned covariance matrices and so unstable inversion and computation of the determinant. There are two major solutions to this problem. First option is to use a regularization method to stabilize the inversion of the covariance matrices. Second option is to use a features extraction/selection method in order to reduce the dimensionality of the samples.

In our study based on GMM, a ridge regularization method described in Section 2.2 is implemented. Then we focus on a feature selection method named sequential forward features selection presented in Section 2.3.2 and an improvement of this algorithm, the sequential floating forward features selection introduced in Section 2.3.2.

In order to evaluate the benefits of such methods for classification, we introduce in the next section several functions called criterion functions.

2.2. Ridge regularization

2.2.1. Ridge regularization

As introduced in the previous section, the aim of a regularization method is to stabilize the inversion of the covariance matrix in the context of high dimensionality which often imply a badly-conditioned matrix.

The ridge regularization is a particular case of the Tikhonov regularization introduced in [12] which aim to penalize the linear problem used to compute the inverse of the covariance matrix by the norm of the inverse. We want to obtain the regularized inverse by solving

$$\hat{\mathbf{A}} = \min_{\mathbf{A}} \|\mathbf{\Sigma A} - \mathbf{I}\|^2 + \|\mathbf{\Gamma A}\|^2. \tag{6}$$

In deriving the expression and setting to zero, we find that the explicit solution to this problem is

$$\hat{\mathbf{A}} = (\mathbf{\Sigma}^t\mathbf{\Sigma} + \mathbf{\Gamma}^t\mathbf{\Gamma})^{-1}\mathbf{\Sigma}. \tag{7}$$

There are then two cases, either we choose $\mathbf{\Gamma} = \sqrt{\tau}\mathbf{I}$ which corresponds to the Tikhonov regularization or $\mathbf{\Gamma} = \sqrt{\tau}\mathbf{\Sigma}^{\frac{1}{2}}$ which corresponds to the ridge regularization with τ a scalar. Then, we get

$$\text{Tikhonov: } \hat{\mathbf{A}} = (\mathbf{\Sigma}^2 + \tau\mathbf{I})^{-1}\mathbf{\Sigma}, \tag{8}$$

$$\text{Ridge: } \hat{\mathbf{A}} = (\mathbf{\Sigma} + \tau\mathbf{I})^{-1}. \tag{9}$$

It is interesting to see the effect of the regularization method through the eigenvalues of the regularized inverse. Thus, we obtain the following expression

$$\text{Tikhonov: } \hat{\lambda}_i^{-1} = \frac{\lambda_i}{\lambda_i^2 + \tau}, \tag{10}$$

$$\text{Ridge: } \hat{\lambda}_i^{-1} = \frac{1}{\lambda_i + \tau}, \tag{11}$$

where λ_i is the ith eigenvalue of the covariance matrix and $\hat{\lambda}_i^{-1}$ the eigenvalue of the regularized inverse.

With the help of Figure 2, we can see that the difference between these two regularization is the way to treat small eigenvalues. The ridge regularization has two advantages. First, it is monotonous and secondly it does not converge to zero when eigenvalue tends to zero.

In particular, the ridge regularization proposes to solve the problem in adding a small value τ to each eigenvalues of the covariance matrix. In our case, we choose to add the same constant for each class but it is not compulsory in a general case.

Thus, in practice, we choose to use ridge regularization. To perform the regularization, the covariance matrix is decomposed with a diagonalization algorithm for symmetric matrices s.t. $\mathbf{\Sigma} = \mathbf{Q\Lambda Q}^t$ where $\mathbf{\Lambda}$ is the diagonal matrix of eigenvalues and $\mathbf{Q}$ is the matrix of associated eigenvectors. When the decomposition is available, the regularization is almost finished and the decision function is simply rewritten as follow

$$Q_c(\mathbf{x}) = -(\mathbf{x} - \boldsymbol{\mu}_c)^t\mathbf{Q}_c(\mathbf{\Lambda}_c + \tau\mathbf{I})^{-1}\mathbf{Q}_c^t(\mathbf{x} - \boldsymbol{\mu}_c) - \log(|\mathbf{\Sigma}_c + \tau\mathbf{I}|) + 2\log(\pi_c) - d\log(2\pi), \tag{12}$$

where $\mathbf{I}$ is the identity matrix. It is to be noticed that $(\mathbf{\Lambda} + \tau\mathbf{I})$ is also a diagonal matrix and so the inverse is easily obtained in inverting each diagonal element of the matrix.

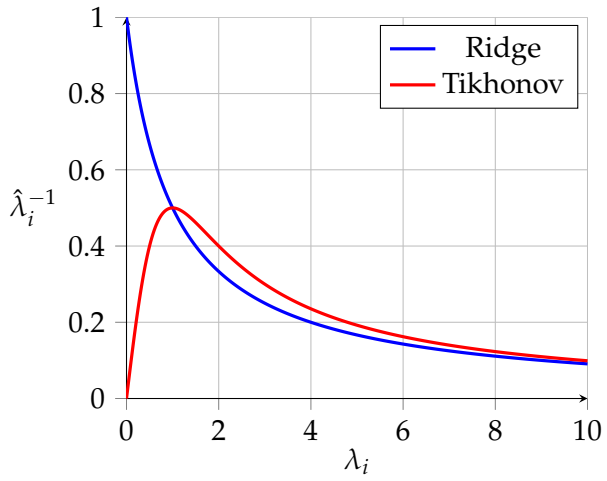


Figure 2: Consequence of regularization on eigenvalues with $\tau = 1$.

2.2.2. Implementation

The main steps of the training algorithm are summarized in Algorithm 1. Indeed, before training the GMM model, a preliminary step is needed in order to determine a good value for the parameter τ. A gridsearch is performed with a set of values $\mathcal{P}$ given by the user. To estimate the quality of the classification with a given τ, one of the good classification criterion presented in Section 2.3.1 is used. As explained at the end of the same section, the classification rate is computed with a cross-validation process.

A particularity of this regularization method is used to accelerate the gridsearch. More precisely, the two computationally costly step are the computation of the inverse and the determinant of the regularized covariance matrices and it is important to see that these operations can be performed only once to test all τ. The covariance matrix is diagonalized before selecting τ and the eigenvalues and eigenvectors are stored and the regularized inverse and determinant can be update as follows

$$\begin{aligned} \hat{\mathbf{\Sigma}}_c^{-1} &= \mathbf{Q}_c(\mathbf{\Lambda}_c + \tau\mathbf{I})^{-1}\mathbf{Q}_c^t, \\ \log(|\hat{\mathbf{\Sigma}}_c|) &= \log(|\mathbf{\Sigma}_c + \tau\mathbf{I}|) = \sum_{i=1}^{d} \log(\lambda_i + \tau). \end{aligned} \tag{13}$$

Finally, another trick could used to optimize the gridsearch. As explained in Section 3.1.1, it is possible to avoid the learning step on the $(k-1)$ fold and instead learn the model with the whole dataset and deduce the submodel used for cross-validation from the complete model and the mean vector and covariance matrix of the k^{th} fold.

As stated before, we choose to implement a basic ridge regression method and only the small improvement explained herebefore as been proposed. To the contrary, we choose in our study to focus more on a feature selection method named sequential forward features selection described in following section.

2.3. Features selection

The objective of feature selection is to select a subset of variables to describe each sample and so to get rid of high dimensionality and its curse.

Features selection algorithm may be divided into three types. The first type which called filter method is based uniquely on data analysis. Features are ranked according to a statistical analysis of the data. For example, the Principal Component Analysis (PCA) described in [4] is a typical filter method or else various metrics [13], [14], [15].

Algorithm 1 Ridge regularization training steps

Require: $\mathcal{S}, k, \mathcal{P}$

Learn GMM with $\mathcal{S}$
Randomly cut $\mathcal{S}$ in k subsets such as $\mathcal{S}_1 \cup ... \cup \mathcal{S}_k = \mathcal{S}$ and $\mathcal{S}_i \cap \mathcal{S}_j = \emptyset$
for all $\mathcal{S}_i$ **do**
  Update GMM model for i^{th} fold with Equations 1 and 2
  Compute decomposition of covariance matrices of each class s.t. $\mathbf{\Sigma}_c = \mathbf{Q}_c \mathbf{\Lambda}_c \mathbf{Q}_c^t$
  for all $\tau \in \mathcal{P}$ **do**
    Update inverse and logdet with Equations 13
    Estimate classification rate on $\mathcal{S}_i$ using Equation 12
  end for
end for
Average the classification rate over the k folds
Compute $\arg\max_\tau$ of the classification rates to get the best parameter τ^*
Parametrize the GMM with τ^*

The second sort are known as wrapper methods which can be seen as search method to determine the best set of variables for a given learning model. As exhaustive searches are out of question in a practical amount of time, numerous search strategies have been designed some optimal under particular hypothesis [16] and other suboptimal but easier to set up [17], [18]. The advantage of such method compare to filter method is that they are dedicated to a particular model but on the other hand, as they require the training of multiple models to test various set of variables, they tend to be slower.

The third type corresponds to the embedded method which do not separate the features selection process from the learning algorithm and allow interaction between the two processes. The basic example of such method is the decision tree algorithm in which a feature is selected for creation of each node. Embedded methods also exist for other model, e.g. SVM ([19], [20]).

The selection method proposed in this work in a wrapper method associated to GMM models. It is important to underline that this method is a *selection* method and not an *extraction* method. It means the subset of variables obtained is composed of actual variables of the original set and not variables built as combination of several others as it is the case for example with PCA. This choice is made to assure an easier interpretation by the user of the obtained subset of variables.

Thus, in order to set a wrapper method are needed a function to rank the various features which we define in Section 2.3.1 and a search strategy. Two search algorithms are presented: the sequential forward features selection method (Section 2.3.2) and the sequential floating forward feature selection method (2.3.2) the second being a more complex variation of the first.

2.3.1. Criterion function

Criterion functions aim to evaluate either a rate of good classification or the separability/similarity of class distributions. These functions are used to estimate which sets of variables are the best to represent data, to assure the separability of the classes and to perform classification. The choice of a criterion function is the choice of a way to compare sets of variables.

Measures of good classification

As described in [9] (chapter 4), all this measures of good classification are based on an error matrix M called confusion matrix which is defined so that M_{ij} is the number of samples of class i classified as class j. The confusion matrix allows the computation of several interesting values relatively to each class:

- the number of True Positive (TP) corresponding to good prediction;
- the number of True Negative (TN) corresponding to good classification of the other class;
- the number of False Negative (FN) corresponding to the samples of the class labeled as an other class;
- the number of False Positive (FP) corresponding to the samples wrongly classified as part of of this class.

Figure 3 illustrates the definition of this values.

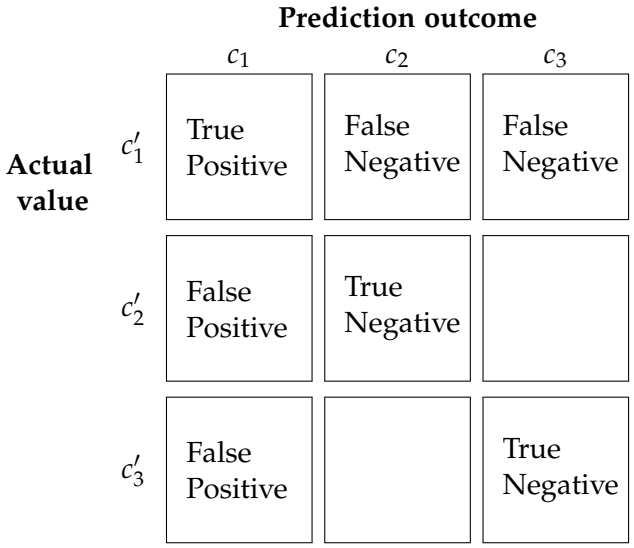


Figure 3: Confusion matrix with TP, TN, FP and FN relatively to c_1

Global indices:

The overall accuracy is simply the rate of the number of samples with the correct predicted label over the number of samples. This metric is easy to interpret but is biased in the case of unbalanced classes.

The Cohen's kappa is a statistic which measure the probabilities of agreement between predictions and ground-truth. This metric tends to give an equal importance to each class.

The mean F1 score is the average of the F1 score for each class and the F1 score is the harmonic mean of the precision (number of True Positive over True Positive plus False Positive) and the recall (number of True Positive over True Positive plus False Negative).

$$\text{Overall Accuracy} = \frac{\text{TP}}{n}. \tag{14}$$

$$\text{Kappa} = \frac{pa - pr}{1 - pr}. \tag{15}$$

$$\text{F1 Mean} = \frac{2\text{TP}}{2\text{TP} + \text{FN} + \text{FP}}. \tag{16}$$

where TP stands for True Positive, FN for False Negative, FP for False Positive, pa is the probability of agreement defined by $pa = Overall Accuracy$ and pr the probability of random agreement defined by $pr = \sum_{c=1}^{C} \frac{TP_c}{FP_c} \frac{TP_c}{FN_c}$.

In order to estimate classification rate, the GMM has to be trained and tested on separate datasets. In our case, we choose to use a cross-validation process over our training set. Thus, the training set is divided in k folds, then, the training is done with $(k-1)$ folds and the performances are estimated with the remaining fold. For more details about cross-validation, see [10] (Chapter 7.10).

Measures of similarity between distributions

The second type of criterion functions is a measure of distance between two distributions. These measures are called divergence function and are defined so that, if we note S a space of all probability distribution with same support, it verifies

$$\forall (p,q) \in S, \mathrm{Div}(p,q) \geq 0,$$
$$\mathrm{Div}(p,q) = 0 \Leftrightarrow p = q.$$

More specifically, we focus on two particular divergence: the Kullback–Leibler divergence and the Jeffries–Matusita distance. The advantage of these divergence is that they have a explicit expression in the case of Gaussian model. The simplification allows us to get rid of any integration calculus which could be a major problem when dealing with high-dimensional data.

The Kullback–Leibler divergence measures the amount of information lost when the first distribution is approximated by the second one. The formal definition is

$$\mathrm{Div}_{KL}(c_i, c_j) = \int_{\mathbf{x}} p(\mathbf{x}|c_i) \ln(\frac{p(\mathbf{x}|c_i)}{p(\mathbf{x}|c_j)}) d\mathbf{x}. \tag{17}$$

And in the case of Gaussian model, it can be rewritten as follows

$$\mathrm{Div}_{KL}(c_i, c_j) = \frac{1}{2}\left(\mathrm{Tr}(\mathbf{\Sigma}_{c_i}^{-1}\mathbf{\Sigma}_{c_j}) + (\boldsymbol{\mu}_{c_i} - \boldsymbol{\mu}_{c_j})^t \mathbf{\Sigma}_{c_i}^{-1}(\boldsymbol{\mu}_{c_i} - \boldsymbol{\mu}_{c_j}) - d + \log\left(\frac{|\mathbf{\Sigma}_{c_i}|}{|\mathbf{\Sigma}_{c_j}|}\right)\right), \tag{18}$$

where Tr is the trace operator and d the dimension of the distribution.

It can be noticed that the KL divergence is not symmetric, i.e. $\mathrm{Div}_{KL}(c_i, c_j) \neq \mathrm{Div}_{KL}(c_j, c_i)$, and so the symmetrized version is used to compute the criterion function. In the case of Gaussian model, the symmetrization induces the following simplification of the formula

$$\begin{aligned}\mathrm{SKL}_{ij} &= \mathrm{Div}_{KL}(c_i, c_j) + \mathrm{Div}_{KL}(c_j, c_i) \\ &= \frac{1}{2}\left(\mathrm{Tr}(\mathbf{\Sigma}_{c_i}^{-1}\mathbf{\Sigma}_{c_j} + \mathbf{\Sigma}_{c_j}^{-1}\mathbf{\Sigma}_{c_i}) + (\boldsymbol{\mu}_{c_i} - \boldsymbol{\mu}_{c_j})^t(\mathbf{\Sigma}_{c_i}^{-1} + \mathbf{\Sigma}_{c_j}^{-1})(\boldsymbol{\mu}_{c_i} - \boldsymbol{\mu}_{c_j}) - 2d\right).\end{aligned} \tag{19}$$

Moreover, the divergence is computed between two classes and to obtain a unique value the weighted mean of divergence measures is taken

$$\mathrm{C}_{SKL} = \sum_{i=1}^{C}\sum_{j=i+1}^{C} \pi_{c_i}\pi_{c_j}\mathrm{SKL}_{ij}. \tag{20}$$

The Bhattacharyya distance is defined as follows

$$\mathrm{B}_{ij} = -\ln\left(\int_{\mathbf{x}} \sqrt{p(\mathbf{x}|c_i)p(\mathbf{x}|c_j)}\, d\mathbf{x}\right). \tag{21}$$

And in the case of Gaussian model:

$$\mathrm{B}_{ij} = \frac{1}{8}(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)^t\left(\frac{\mathbf{\Sigma}_i + \mathbf{\Sigma}_j}{2}\right)^{-1}(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j) + \frac{1}{2}\ln\left(\frac{|\mathbf{\Sigma}_i + \mathbf{\Sigma}_j|}{\sqrt{|\mathbf{\Sigma}_i||\mathbf{\Sigma}_j|}}\right). \tag{22}$$

The Jeffries–Matusita distance is a measure based on the Bhattacharyya distance but transform in a way that the distance saturate if the separability between the two distribution increases. The JM distance is defined according to

$$\mathrm{JM}_{ij} = \sqrt{\int_{\mathbf{x}} \left[\sqrt{p(\mathbf{x}|c_i)} - \sqrt{p(\mathbf{x}|c_j)} \right]^2 d\mathbf{x}}. \tag{23}$$

And the Jeffries–Matusita distance can be rewritten according to the Bhattacharyya distance

$$\mathrm{JM}_{ij} = \sqrt{2\{1 - \exp[-B_{ij}]\}}. \tag{24}$$

As for the KL divergence, a weighted mean of the distance between two classes is computed to aggregate the measures in a single value.

According to [11], it is interesting to notice that the KL divergence increases quadratically with respect to the distance between the mean vectors of the class distributions whereas the measures of good classification we used asymptotically tends to one when distributions are perfectly separable. On the contrary, the JM distance tends to saturate as these measures of good classification.

2.3.2. Selection method

Sequential forward features selection The Sequential Forward Selection (SFS) starts with an empty set of selected features. Then it tests at each step for all the remaining features the value of a criterion function J chosen among the ones presented in Section 2.3.1 when the feature is added to the pool of selected features. The feature that maximizes the criterion function is definitively added to the pool of selected features and it moves to the next iteration. The algorithm stops when a given number of variables *maxVarNb* has been selected. The Algorithm 2 presents the process in details.

To summarize, the features are selected one by one and the set of selected features are never questioned. It results in a reasonable computational time but a suboptimal solution to the selection problem meaning that a better subset of variables could exist.

Algorithm 2 Sequential forward features selection

Require: $\mathcal{S}, J, \mathrm{maxVarNb}$
$S = \emptyset$
$F = \{\text{all variables}\}$
while $\mathrm{card}(S) \leq maxVarNb$ **do**
  for all $f_i \in F$ **do**
    $R_i = J(\{S + f_i\})$
  end for
  $j = \arg\max_i R_i$
  $S = \{S + f_j\}$
  $F = F \setminus f_j$
end while
return S

Figure 4 shows a toy dataset with 2 classes and 2 features. The projection of the Gaussian distribution learned for each class can be visualized on the corresponding axis. In this examples, each class has been generated given a Gaussian law.

In the cases of the example presented in Figure 4, we compute the criterion functions described in 2.3.1 and summarize it in Table 1. We can see that all the criterion functions are maximum with the feature 1 which makes sense because the projection of the learned Gaussian distribution effectively seems separable only for feature 1. So the algorithm will has expected select the feature 1 as the best

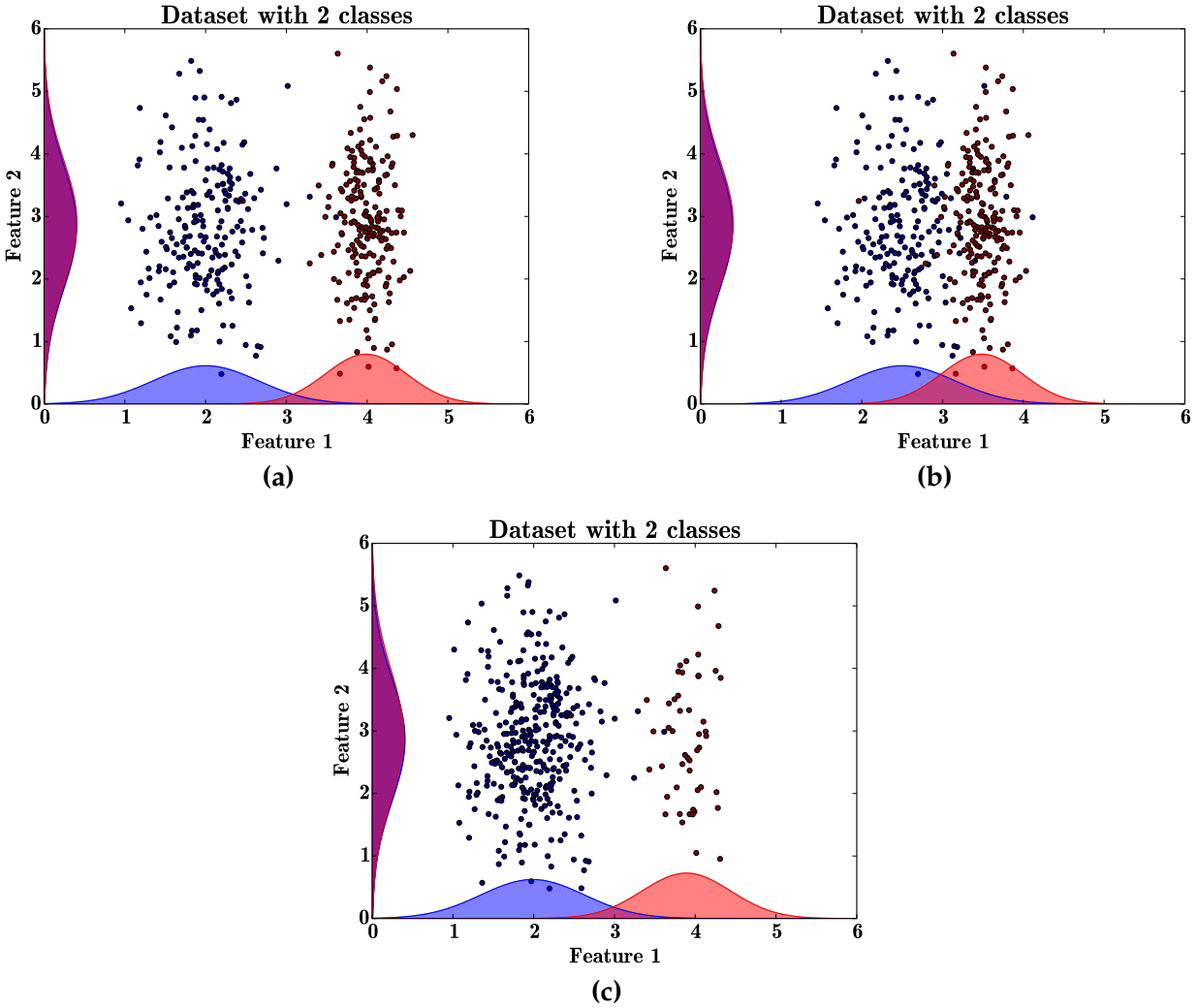


Figure 4: Example datasets: **(a)** 2 classes and 2 features but only one informative feature, **(b)** same dataset with a reduced gap between means of informative feature, **(c)** similar dataset with unbalanced classes (350/50 samples).

features to perform classification. The case (c) also shows that the measures are robust to unbalanced classes except the overall accuracy which, as we can see, do not make much difference between the two features because of the imbalance.

	Case a			Case b			Case c		
	Feat 1	Feat 2	Feat 1+2	Feat 1	Feat 2	Feat 1+2	Feat 1	Feat 2	Feat 1+2
Overall Accuracy	0.995	0.445	0.995	0.935	0.445	0.925	0.985	0.87	0.985
Cohen's kappa	0.99	-0.11	0.99	0.87	-0.11	0.85	0.95	0	0.95
F1-score mean	0.995	0.43	0.995	0.935	0.43	0.92	0.97	0.47	0.97
KL divergence	13.04	0.25	13.44	3.48	0.25	3.77	2.66	0.12	2.79
Jeffrey-Matusita distance	0.35	0.19	0.35	0.31	0.19	0.32	0.16	0.09	0.16

Table 1: Criterion functions values corresponding to Figure 4 dataset (50% for training/50% for testing).

Sequential floating forward feature selection The Sequential Floating Forward Selection (SFFS) is actually based on two algorithms: the SFS described above and the Sequential Backward Selection (SBS). The SBS is the backward equivalent of SFS. The difference is that it starts with every features in the pool of selected features and tries at each step to remove the less significant one in term of the given criterion function.

The SFFS works as the SFS but between each step of the SFS algorithm a backward selection is operated. At the end of the SBS step, the value of the criterion function is compared to the best value ever obtained with a set of features of the same size and if the new value is better the feature puts into question is effectively taken away and the next step is again a SBS but if the new value is not better the SBS step is forgotten and it moves to the next SFS step. The algorithm stops when a given number of features has been selected. The Algorithm 3 sums up the method.

This SFFS algorithm eventually tests more solutions than the SFS algorithm. The results are expected to be better but the trade-off is an increased computational time which is dependent on the complexity of the dataset.

Algorithm 3 Sequential floating forward features selection

Require: $\mathcal{S}, J, \text{maxVar}$
$S = \emptyset$
$F = \{\text{all variables}\}$
$k = 0$
while Repeat till S contained maxVar variables **do**
  for all $f_i \in F$ **do**
    $R_i = J(\{S(k) + f_i\})$
  end for
  $j = \arg\max_i R_i$
  if $R_j \geq J(S^{(k+1)})$ **then**
    $k = k + 1$
  else
    $S^{(k+1)} = \{S(k) + f_i\}$
    $k = k + 1$
    $\text{flag} = 1$
    while $k > 2$ and $\text{flag} = 1$ **do**
      for all $f_i \in S^{(k)}$ **do**
        $R_i = J(\{S(k) \setminus f_i\})$
      end for
      $j = \arg\max_i R_i$
      if $R_j < J(S^{(k-1)})$ **then**
        $S^{(k-1)} = \{S(k) \setminus f_i\}$
        $k = k - 1$
      else
        $\text{flag} = 0$
      end if
    end while
  end if
end while
return S

3. Smart implementation

3.1. Statistical update rules

A major contribution of our work is the optimization in term of computational efficiency. More precisely, three steps of our method has been upgraded in order to reduce computational time. First of all, the GMM model is learned only once using samples. When a covariance matrix or a mean vector of a reduced set of variables is required, we get it from the global model learned at the beginning.

Secondly, when a cross-validation is performed, we do not learn a submodel, i.e. a GMM model trained with $(n-1)$ folds, using the $(n-1)$ folds but, instead, we derive it from the global model and the covariance matrices and mean vectors of the fold used for validation for this given submodel. The process is described in details in Section 3.1.1.

Finally and most importantly, when variables are tested one by one during a selection step, costly operations are made when computing criterion functions for each variable especially the computation of the inverse of covariance matrices and its determinant. We manage to set up several update rules which allow us to compute this inverse and determinant only once to test all variables. These update rules are presented in Section 3.1.2.

3.1.1. Update for cross validation

Based on [7], a method to accelerate the k-fold cross-validation process in the case of criterion functions based on correct classification measures was implemented. The idea is to estimate the GMM model with the whole training set and then, instead of training a model on $(k-1)$ folds, the complete model and the mean vector and the covariance matrix of the k^{th} fold is used to derive the corresponding submodel.

The following formulae can be obtained (details of calculation in Appendix B)

Proposition 1. *(CV mean update)*

$$\boldsymbol{\mu}_c^{n_c-\nu_c} = \frac{n_c\boldsymbol{\mu}_c^{n_c} - \nu_c\boldsymbol{\mu}_c^{\nu_c}}{n_c-\nu_c}$$

Proposition 2. *(CV covariance matrix update)*

$$\boldsymbol{\Sigma}_c^{n_c-\nu_c} = \frac{1}{n_c-\nu_c-1}((n_c-1)\boldsymbol{\Sigma}_c^{n_c} - (\nu_c-1)\boldsymbol{\Sigma}_c^{\nu_c} - \frac{n_c\nu_c}{(n_c-\nu_c)}(\boldsymbol{\mu}_c^{\nu_c}-\boldsymbol{\mu}_c^{n_c})(\boldsymbol{\mu}_c^{\nu_c}-\boldsymbol{\mu}_c^{n_c})^t)$$

where ν_c is the number of samples of class c removed from the set, $\boldsymbol{\mu}_c^{n_c}$ the mean vector estimated with this ν_c samples and $\boldsymbol{\Sigma}_c^{\nu_c}$ the covariance matrix estimated with this ν_c samples.

Additionally, the update of the GMM model is the occasion to precompute and store the constant $2\log(\pi_c)$ which is a term of the decision function constant for a given training set.

3.1.2. Criterion function computation

As explained earlier, at each iteration the SFS and SFFS algorithms compute the value of a criterion function for every possible set of features composed of the selected ones augmented by one of the remaining features. One of the main achievements of this work is to reduce the computational time needed to compute the criterion function of augmented sets.

We note $\boldsymbol{\Sigma}^{(k-1)}$ a covariance matrix of the $(k-1)^{th}$ iteration, i.e., a covariance of the selected features and $\boldsymbol{\Sigma}^{(k)}$ the corresponding covariance matrix at the k^{th} iteration, i.e., the covariance of the augmented set. Then, the inverse of the covariance matrix $(\boldsymbol{\Sigma}^{(k)})^{-1}$, the quadratical term $(\mathbf{x}^{(k)})^t(\boldsymbol{\Sigma}^{(k)})^{-1}\mathbf{x}^{(k)}$ and the determinant $\log|\boldsymbol{\Sigma}^{(k)}|$ can be expressed in function of terms of the $(k-1)^{th}$ iteration. These calculi use the fact that the covariance matrix is a positive definite symmetric matrix to simplify formulae using matrices defined by blocks. These update rules are summed up hereafter and are available in [21] (chapter 9.2).

In order to easily understand which terms are available and which ones needs to be computed, we choose to set in green in the update rules the terms to already computed.

As $\boldsymbol{\Sigma}^{(k)}$ is a positive definite symmetric matrix, we can use the following notation

$$\boldsymbol{\Sigma}^{(k)} = \begin{bmatrix} \boldsymbol{\Sigma}^{(k-1)} & \mathbf{u} \\ \mathbf{u}^t & \sigma_{kk} \end{bmatrix},$$

where $\mathbf{u}$ is the k^{th} column of the matrix without the diagonal element i.e. $\mathbf{u}_i = \mathbf{\Sigma}^{(k)}_{i,k}$ with $i \in [1, k-1]$.

Using the formula of the inverse of a block matrix, the following formula expressing $(\mathbf{\Sigma}^{(k)})^{-1}$ in function of $(\mathbf{\Sigma}^{(k-1)})^{-1}$ is obtained

Proposition 3. *(Forward update rule for inverse of covariance matrix)*

$$\boxed{(\mathbf{\Sigma}^{(k)})^{-1} = \begin{bmatrix} (\mathbf{\Sigma}^{(k-1)})^{-1} + \frac{1}{\alpha}(\mathbf{\Sigma}^{(k-1)})^{-1}\mathbf{u}\mathbf{u}^t(\mathbf{\Sigma}^{(k-1)})^{-1} & -\frac{1}{\alpha}(\mathbf{\Sigma}^{(k-1)})^{-1}\mathbf{u} \\ -\frac{1}{\alpha}\mathbf{u}^t(\mathbf{\Sigma}^{(k-1)})^{-1} & \frac{1}{\alpha} \end{bmatrix}}$$

where $\alpha = \sigma_{kk} - \mathbf{u}^t(\mathbf{\Sigma}^{(k-1)})^{-1}\mathbf{u}$.

To insist again on the advantage given by this update rules, it means that, if we want to select a variable among one hundred, instead of computing one hundred times $(\mathbf{\Sigma}^{(k)})^{-1}$, we will compute $(\mathbf{\Sigma}^{(k-1)})^{-1}$ once and then with a few matrix multiplications and sums we can get the hundred inverse and so save a lot of processing time.

In the case of a backward step in SFFS algorithm, we need to invert the formula in order to compute $(\mathbf{\Sigma}^{(k-1)})^{-1}$ knowing $(\mathbf{\Sigma}^{(k)})^{-1}$. So if we note

$$(\mathbf{\Sigma}^{(k)})^{-1} = \begin{bmatrix} \mathbf{A} & \mathbf{v} \\ \mathbf{v}^t & \frac{1}{\alpha} \end{bmatrix}.$$

Thus, the update rule becomes

Proposition 4. *(Backward update rule for inverse of covariance matrix)*

$$\boxed{(\mathbf{\Sigma}^{(k-1)})^{-1} = \mathbf{A} - \alpha\mathbf{v}\mathbf{v}^t}$$

Proof.

$$\begin{aligned} \mathbf{A} - \alpha\mathbf{v}\mathbf{v}^t &= (\mathbf{\Sigma}^{(k-1)})^{-1} + \frac{1}{\alpha}(\mathbf{\Sigma}^{(k-1)})^{-1}\mathbf{u}\mathbf{u}^t(\mathbf{\Sigma}^{(k-1)})^{-1} - \alpha(-\frac{1}{\alpha}(\mathbf{\Sigma}^{(k-1)})^{-1}\mathbf{u})(-\frac{1}{\alpha}\mathbf{u}^t(\mathbf{\Sigma}^{(k-1)})^{-1})) \\ &= (\mathbf{\Sigma}^{(k-1)})^{-1} \end{aligned}$$

□

A formula can also be deduced for the quadratical term of the decision function. If we note $(\mathbf{x}^{(k)})^t = \begin{bmatrix} (\mathbf{x}^{(k-1)})^t & x^k \end{bmatrix}$, we get the following proposition

Proposition 5. *(Update rule for quadratical term)*

$$\boxed{(\mathbf{x}^{(k)})^t(\mathbf{\Sigma}^{(k)})^{-1}\mathbf{x}^{(k)} = (\mathbf{x}^{(k-1)})^t(\mathbf{\Sigma}^{(k-1)})^{-1}\mathbf{x}^{(k-1)} + \alpha(\begin{bmatrix} \mathbf{v}^t & \frac{1}{\alpha} \end{bmatrix}\mathbf{x}^{(k)})^2}$$

Proof.

$$
\begin{aligned}
(\mathbf{x}^{(k)})^t(\mathbf{\Sigma}^{(k)})^{-1}\mathbf{x}^{(k)} &= \begin{bmatrix} (\mathbf{x}^{(k-1)})^t & x_k \end{bmatrix} \begin{bmatrix} \mathbf{A} & \mathbf{v} \\ \mathbf{v}^t & \frac{1}{\alpha} \end{bmatrix} \begin{bmatrix} \mathbf{v} \\ x_k \end{bmatrix} \\
&= \begin{bmatrix} (\mathbf{x}^{(k-1)})^t\mathbf{A} + x_k\mathbf{v}^t & (\mathbf{x}^{(k-1)})^t\mathbf{v} + \frac{x_k}{\alpha} \end{bmatrix} \begin{bmatrix} \mathbf{v} \\ x_k \end{bmatrix} \\
&= (\mathbf{x}^{(k-1)})^t\mathbf{A}\mathbf{x}^{(k-1)} + x_k\mathbf{v}^t\mathbf{x}^{(k-1)} + (\mathbf{x}^{(k-1)})^t\mathbf{v}x_k + \frac{(x_k)^2}{\alpha} \\
&= (\mathbf{x}^{(k-1)})^t((\mathbf{\Sigma}^{(k-1)})^{-1} + \frac{1}{\alpha}(\mathbf{\Sigma}^{(k-1)})^{-1}\mathbf{u}\mathbf{u}^t(\mathbf{\Sigma}^{(k-1)})^{-1})\mathbf{x}^{(k-1)} + 2x_k\mathbf{v}^t\mathbf{x}^{(k-1)} + \frac{(x_k)^2}{\alpha} \\
&= (\mathbf{x}^{(k-1)})^t((\mathbf{\Sigma}^{(k-1)})^{-1} + \alpha\mathbf{v}\mathbf{v}^t)\mathbf{x}^{(k-1)} + 2x_k\mathbf{v}^t\mathbf{x}^{(k-1)} + \frac{(x_k)^2}{\alpha} \\
&= (\mathbf{x}^{(k-1)})^t(\mathbf{\Sigma}^{(k-1)})^{-1}\mathbf{x}^{(k-1)} + \alpha((\mathbf{x}^{(k-1)})^t\mathbf{v}\mathbf{v}^t\mathbf{x}^{(k-1)} + 2\frac{x_k}{\alpha}\mathbf{v}^t\mathbf{x}^{(k-1)} + \frac{(x_k)^2}{\alpha^2}) \\
&= (\mathbf{x}^{(k-1)})^t(\mathbf{\Sigma}^{(k-1)})^{-1}\mathbf{x}^{(k-1)} + \alpha((\mathbf{x}^{(k-1)})^t\mathbf{v} + \frac{x_k}{\alpha})^2 \\
&= (\mathbf{x}^{(k-1)})^t(\mathbf{\Sigma}^{(k-1)})^{-1}\mathbf{x}^{(k-1)} + \alpha(\begin{bmatrix} \mathbf{v}^t & \frac{1}{\alpha} \end{bmatrix}\mathbf{x}^{(k)})^2
\end{aligned}
$$

□

Finally, using the formula of the determinant of a block matrix and after taking the log, we obtain

Proposition 6. *(Update rule for logdet)*

$$\boxed{\log\left(|\mathbf{\Sigma}^{(k)}|\right) = \log\left(|\mathbf{\Sigma}^{(k-1)}|\right) + \log\alpha}$$

Now using all update rules, criterion functions can be rewritten as follows

$$
\begin{aligned}
Q(\mathbf{x}) = &- (\mathbf{x}^{(k-1)} - \boldsymbol{\mu}^{(k-1)})^t(\mathbf{\Sigma}^{(k-1)})^{-1}(\mathbf{x}^{(k-1)} - \boldsymbol{\mu}^{(k-1)}) \\
&- \alpha\left(\begin{bmatrix} \mathbf{v}^t & \frac{1}{\alpha} \end{bmatrix}(\mathbf{x}^{(k)} - \boldsymbol{\mu}^{(k)})\right)^2 \\
&- \log\left(|\mathbf{\Sigma}^{(k-1)}|\right) - \log\alpha + 2\log(\pi_c) + k\log(2\pi);
\end{aligned} \tag{25}
$$

$$
\begin{aligned}
\mathrm{SKL}_{ij} = &\frac{1}{2}\Bigg(\mathrm{Tr}\left((\mathbf{\Sigma}_{c_i}^{(k)})^{-1}\mathbf{\Sigma}_{c_j}^{(k)} + (\mathbf{\Sigma}_{c_j}^{(k)})^{-1}\mathbf{\Sigma}_{c_i}^{(k)}\right) + (\boldsymbol{\mu}_{c_i}^{(k-1)} - \boldsymbol{\mu}_{c_j}^{(k-1)})^t((\mathbf{\Sigma}_{c_i}^{(k-1)})^{-1} + (\mathbf{\Sigma}_{c_j}^{(k-1)})^{-1})(\boldsymbol{\mu}_{c_i}^{(k-1)} - \boldsymbol{\mu}_{c_j}^{(k-1)}) \\
&+ \alpha(\begin{bmatrix} \mathbf{v_i}^t & \frac{1}{\alpha_i} \end{bmatrix}(\boldsymbol{\mu}_{c_i}^{(k)} - \boldsymbol{\mu}_{c_j}^{(k)}))^2 + \alpha(\begin{bmatrix} \mathbf{v_j}^t & \frac{1}{\alpha_j} \end{bmatrix}(\boldsymbol{\mu}_{c_i}^{(k)} - \boldsymbol{\mu}_{c_j}^{(k)}))^2 - 2k\Bigg),
\end{aligned} \tag{26}
$$

with $(\mathbf{\Sigma}_{c_i}^{(k)})^{-1}$ computed with Proposition 3;

$$
\begin{aligned}
\mathrm{B}_{ij} = &\frac{1}{4}(\boldsymbol{\mu}_i^{(k-1)} - \boldsymbol{\mu}_j^{(k-1)})^t(\tilde{\mathbf{\Sigma}}^{(k-1)})^{-1}(\boldsymbol{\mu}_i^{(k-1)} - \boldsymbol{\mu}_j^{(k-1)}) + \frac{1}{4}\tilde{\alpha}(\begin{bmatrix} \tilde{\mathbf{v}}^t & \frac{1}{\tilde{\alpha}} \end{bmatrix}(\boldsymbol{\mu}_i^{(k)} - \boldsymbol{\mu}_j^{(k)}))^2 \\
&+ \frac{1}{2}\ln\left(\frac{|\tilde{\mathbf{\Sigma}}^{(k-1)}|}{\sqrt{|\mathbf{\Sigma}_i^{(k-1)}||\mathbf{\Sigma}_j^{(k-1)}|}}\right) + \frac{1}{2}\ln\left(\frac{\tilde{\alpha}}{\sqrt{\alpha_i\alpha_j}}\right),
\end{aligned} \tag{27}
$$

where $\tilde{\mathbf{\Sigma}} = \mathbf{\Sigma}_i + \mathbf{\Sigma}_j$.

It can be noticed that these new formulae allow us to precompute the computationally heavy terms because they are the same for all possible augmented sets at a given iteration of the selection. The Algorithm 4 illustrates the optimization of the Algorithm 3.

3.1.3. Computational issues

During an iteration of the selection algorithm, it still needs to invert the covariance matrix of the previous iteration. In order to so, we choose to perform a decomposition in eigenvalues and eigenvectors using algorithm specific for symmetric matrices. In other words, we find $\mathbf{\Lambda}$ the diagonal matrix of eigenvalues and $\mathbf{Q}$ the matrix of eigenvectors s.t. $\mathbf{\Sigma} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^t$.

This decomposition has several advantages. First, it is very simple to compute the determinant once the eigenvalues are available because the determinant is equal to the product of these values. Secondly, it is possible to check the eigenvalues and so to assure a better computational stability. Due to the curse of dimensionality, the covariance matrix is sometimes badly conditioned and the results is that some eigenvalues are really small (below computational precision) and the decomposition helps us to check their actual values and if they are below EPS_FLT which is the machine maximum precision for a float, we set these eigenvalues to EPS_FLT. For the same reason, the constant α used in update rules is also thresholded to EPS_FLT. Algorithm 4 details when computational stability is checked. We choose to underline the steps added compared to Algorithm 3.

Algorithm 4 Sequential floating forward features selection with updates

Require: $\mathcal{S}, J, \text{maxVar}$
$S = \emptyset$
$F = \{\text{all variables}\}$
$k = 0$
while Repeat till S contained maxVar variables **do**
  Diagonalize $\mathbf{\Sigma}^{(k-1)} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^t$
  for all λ_i **do** $\lambda_i = \min(\textbf{EPS\_FLT}, \lambda_i)$
  Precompute $(\mathbf{\Sigma}^{(k-1)})^{-1}$, $(\mathbf{x}^{(k-1)} - \boldsymbol{\mu}^{(k-1)})^t(\mathbf{\Sigma}^{(k-1)})^{-1}(\mathbf{x}^{(k-1)} - \boldsymbol{\mu}^{(k-1)})$ and $\log\left(|\mathbf{\Sigma}^{(k-1)}|\right)$ using Propositions 3, 5 and 6
  for all $f_i \in F$ **do**
    Compute update constant α
    $\alpha = \min(\text{EPS\_FLT}, \alpha)$
    $R_i = J(\{S(k) + f_i\})$ using Equations 25, 26 or 27
  end for
  $j = \arg\max_i R_i$
  if $R_j \geq J(S^{(k+1)})$ **then**
    $k = k + 1$
  else
    $S^{(k+1)} = \{S(k) + f_i\}$
    $k = k + 1$
    flag $= 1$
    while $k > 2$ and flag $= 1$ **do**
      Diagonalize $\mathbf{\Sigma}^{(k-1)} == \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^t$
      for all λ_i **do** $\lambda_i = \min(\textbf{EPS\_FLT}, \lambda_i)$
      Precompute $(\mathbf{\Sigma}^{(k-1)})^{-1}$, $(\mathbf{x}^{(k-1)} - \boldsymbol{\mu}^{(k-1)})^t(\mathbf{\Sigma}^{(k-1)})^{-1}(\mathbf{x}^{(k-1)} - \boldsymbol{\mu}^{(k-1)})$ and $\log\left(|\mathbf{\Sigma}^{(k-1)}|\right)$ using Propositions 3, 5 and 6
      for all $f_i \in S^{(k)}$ **do**
        Compute update constant α
        $\alpha = \min(\text{EPS\_FLT}, \alpha)$
        $R_i = J(\{S(k) \setminus f_i\})$ using Equations 25, 26 or 27
      end for
      $j = \arg\max_i R_i$
      if $R_j < J(S^{(k-1)})$ **then**
        $S^{(k-1)} = \{S(k) \setminus f_i\}$
        $k = k - 1$
      else
        flag $= 0$
      end if
    end while
  end if
end while
return S

3.2. Implementation of the Orfeo Toolbox module

After the calibration of the algorithm with a Python implementation, we aim to realize a C++ implementation of this selection method. And, to make it easily available, we choose to develop a remote module for the Orfeo Toolbox (OTB) designed by the CNES. The current section describes this C++ implementation.

3.2.1. Integration in Orfeo Toolbox

The Orfeo Toolbox is an open-source library for remote sensing image processing. This library allows developers to code external modules and make them available to the community. It is to note that the toolbox already implements several classifiers inherited mostly from OpenCV. Nevertheless, the GMM classifier currently in place is not robust to high dimensionality (see Section 4.2) and that's why we choose to implement our own model.

The module we developed is available on Github [5] and is actually a fork of the template for remote module furnished by OTB developers.

3.2.2. Code structure

Even if a GMM classifier inherited from Opencv library is already available in the OTB, we choose to implement our own version of a GMM classifier in a class named *GMMMachineLearningModel* in order to assure better performance in term of computational time. The second step is to implement a subclass of the GMM classifier including the features selection method.

The *GMMMachineLearningModel* class inherits from the OTB class *MachineLearningModel* which is a basic class used for all classifier in the OTB. The *MachineLearningModel* class enables the management of a list of samples used for training and also declares the virtual methods inherited: *Train()*, *Predict()*, *Save()*, *Load()*, *CanReadFile()*, *CanWriteFile()*.

The *GMMMachineLearningModel* class implements all these virtual methods. The *Train()* aims to train the classifier using a list of samples set as a member of the class. Then, it is possible to use the method *Predict()* to classify a sample and optional get the confidence in the prediction. The methods *Save()* and *Load()* obviously are used to save and load a trained model. And, finally, *CanReadFile()* and *CanWriteFile()* state the validity of a GMM model file.

In addition to these inherited methods, two noticeable methods are implemented. The *Decomposition()* method perform a decomposition of a symmetric matrix in eigenvalues and eigenvectors using a function of the VNL library. It is to notice that this method check the value of the extracted eigenvalues and minor them to EPSILON_FLT (or EPSILON_DBL) which corresponds to computational precision. The idea is to limit computational instability when parameters are badly-estimated due to an insufficient amount of training samples. The second interesting method is *TrainTau()* which concerns the ridge regularization. Given a set of possible regularization constant τ, this method selects the most efficient value in estimating classification rate with cross-validation.

In term of members, the *GMMMachineLearningModel* includes:

- 6 variables describing the model: the number of class, the number of features, the proportion of each class in the training set, the number of samples in each class, the mean vectors of each class and finally the covariance matrices of each class;
- 1 optional meta-parameter: the regularization value τ for ridge regularization;
- 1 variable the result of tau selection (classification rate for each tau);
- 4 precomputed terms used for prediction: eigenvalues of covariance matrices, eigenvectors of covariance matrices, 2 other terms.

[5] https://github.com/Laadr/otbExternalFastFeaturesSelection

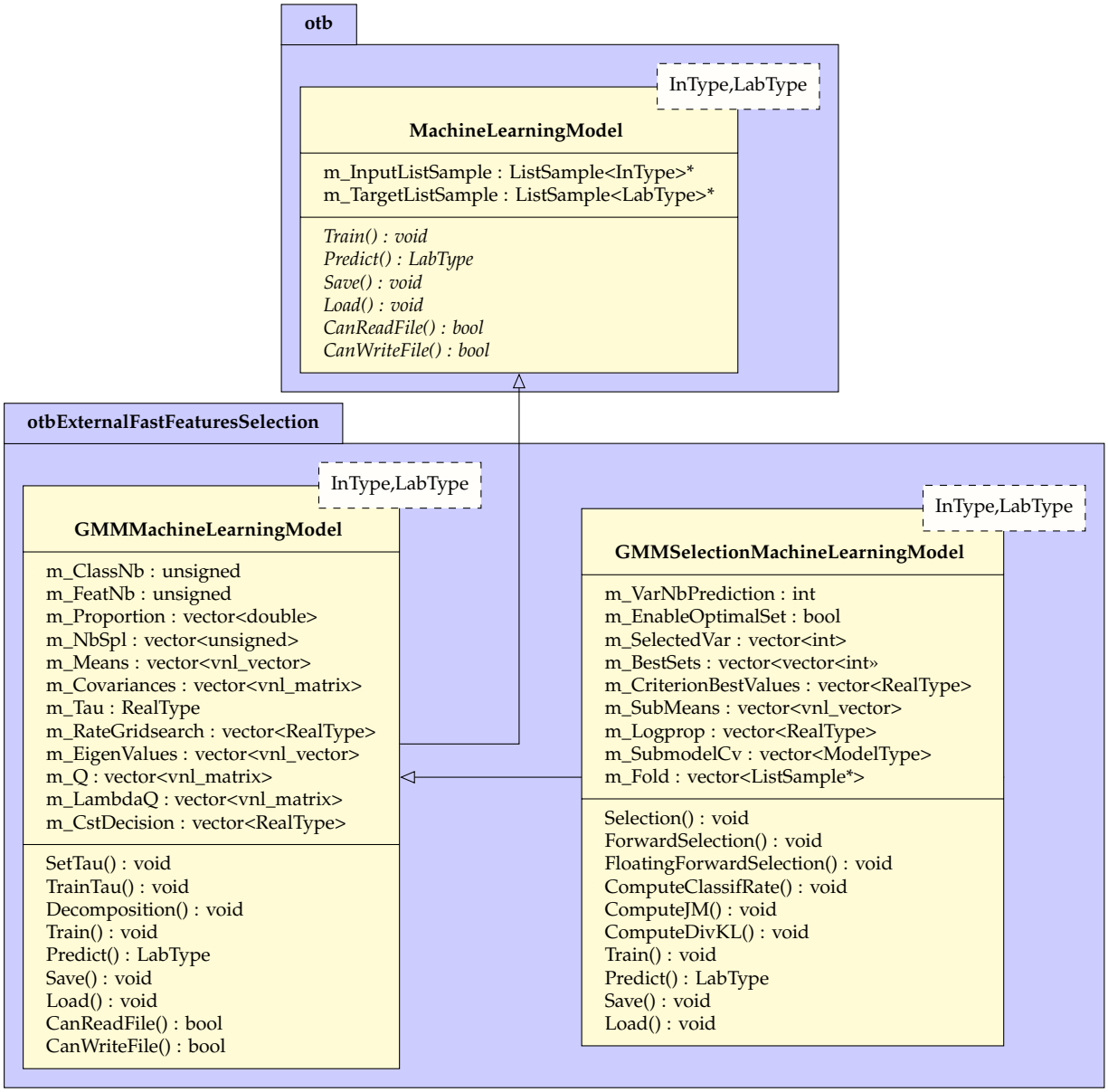


Figure 5: UML digram of OTB remote module.

To implement the selection algorithm, a new class *GMMSelectionMachineLearningModel* inheriting from *GMMMachineLearningModel* is defined. This class keeps the same six methods inherited from *MachineLearningModel* but reimplements *Predict()* to predict with a reduced set of features and *Save()/Load()* to use a second file to handle the results of the selection.

The other methods available in class *GMMSelectionMachineLearningModel* are used to perform the selection algorithms presented in Section 2. The method *Selection()* aims to set the various parameter of the selection and then calls either the *ForwardSelection()* method to use forward feature selection or the *FloatingForwardSelection()* method to use SFFS. Then, 3 different methods are used to evaluate criterion functions during selection. Given a pool of variables available for selection, these functions evaluate the criterion functions for all the possible augmented set. The *ComputeJM()* method is used for Jeffries-Matusita distance, *ComputeDivKL()* for Kullback–Leibler divergence and *ComputeClassifRate()* for the three good classification criteria. It is to notice that the good classification criteria are evaluated using cross-validation.

Several new members are defined in the *GMMSelectionMachineLearningModel* class:

- 2 meta-parameters: one to specify the number of variables to use for prediction and one to enable the process to automatically set the previous meta-parameter to the number maximizing the criterion function;
- 3 variables describing the results of selection: the set to use for prediction, the evolution of criterion function at each iteration and the best sets for each iteration;
- 2 variables used for cross-validation: the set of submodels and the folds of samples;
- 2 precomputed terms used for prediction.

All the code structure is summarized in the UML diagram displayed in Figure 5.

3.2.3. Applications

In order to provide an easy way to use the developed algorithm, we finally develop OTB application. These application allows the user to use and parametrize the algorithm from command line. Three applications are built.

The application *otbcli_TrainGMMApp* create and train a model from class *GMMMachineLearningModel* and the application *otbcli_TrainGMMSelectionApp* same for a model of class *GMMSelectionMachineLearningModel*. These two applications use a raster image and shapefile with groundtruth as input and create a model file.

The last application *otbcli_PredictGMMApp* takes a raster image as input and a model file and perform classification of the image. It generates a classification map in an image file and optionally a confidence map. An other application is available in the OTB to compute classification rate from the image and a groundtruth file.

4. Experimental results

4.1. Test: GMM algorithm variations

In this section, we discuss the results of the features selection method. The forward selection and the floating forward selection are compared and the results with the different criterion function are also discussed. All the experimentation has been conducted with a dataset presented in the next subsection.

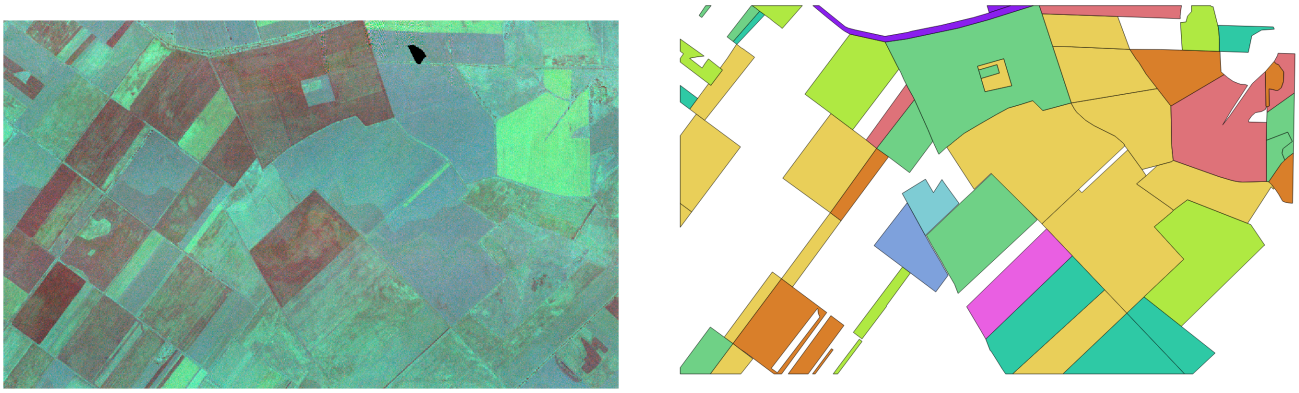
4.1.1. Aisa dataset

The Aisa dataset has been acquired by the AISA Eagle sensor during a flight campaign over Heves, Hungary. It contains 252 bands ranging from 395 to 975 nm. 16 classes have been defined for a total of 361,971 referenced pixels. Figure 6 shows one channel of the multispectral image and the shapefile representing groundtruth and Table 2 presents the repartition of the various classes.

4.1.2. Experimental results

For the results discussed in this section, the selection process was repeated 20 times with each time a different training set and the classification rate and time processing presented hereafter are the means over these 20 trials. We choose to present the results using Cohen's kappa but the same results have been obtained when we look at the mean of F1-scores.

Figure 7 corresponds to the evolution of the classification rate expressed with Cohen's kappa in function of the number of selected variables. We observe that at first the classification rate rapidly increase before stabilizing and then slowly decrease. This is the typical behavior expected when using features selection and a GMM classifier and it illustrated the Hughes phenomenon [22] which



(a) **(b)**

Figure 6: Aisa dataset: **(a)** a single band of the image, **(b)** groundtruth.

Class	Number of samples
Winter wheat	136,524
Sunflower	61,517
Green fallow last year treatment	30,197
Alfalfa	17,626
Maize	18,278
Millet	7,199
Broadleaved forest	10,746
Meadow	23,283
Winter barley	2,799
Reed	4,222
Water course	4,773
Rape	26,566
Green fallow with shrub	9,272
Green fallow last year treated	3,426
Pasture	2,107
Oat	3,436

Table 2: table
Repartition of classes in Aisa dataset.

states that with a given number of samples, prediction accuracy first increases with dimensionality but then decays when the complexity is higher than some optimum value. In our particular case, $\frac{d(d+3)}{2}$ parameters have to be estimated. With $d = 13$, it means 104 parameters and we use in this example 100 samples by class. Figure 7 confirms this theoretical limit, as we can see, when the number of features is higher than 13, the classification rate actually becomes to decrease. In practice, we want to use the number of variables which maximizes the classification rate even if the maximum is not always easily found for example because of local maxima.

It is also interesting to confirm the influence of the size of the training set. From Figure 8, we see that, either with an equalized training set or with a proportional training set, the classification rate is obviously better with more samples and also that the maximum is reached with more selected variables which means that more samples allows to evaluate more precisely the relevant variables.

Using Figure 9, we can compare the results when using forward selection or SFFS. And we see

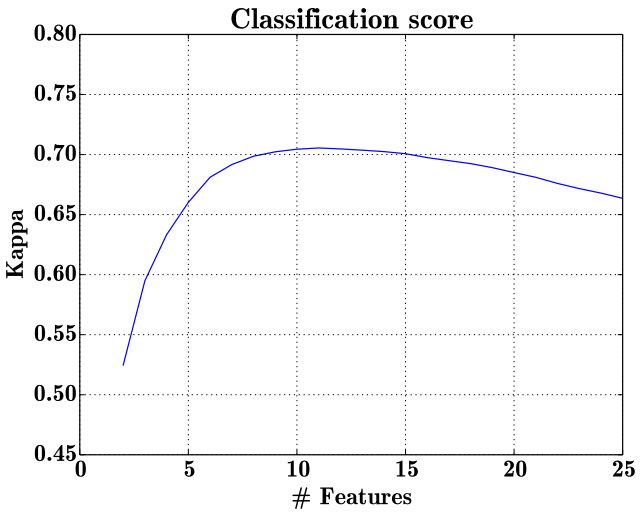


Figure 7: Classification results averaged on 20 trials using forward selection and Jeffries-Matusita distance as criterion with 100 samples by class in training set.

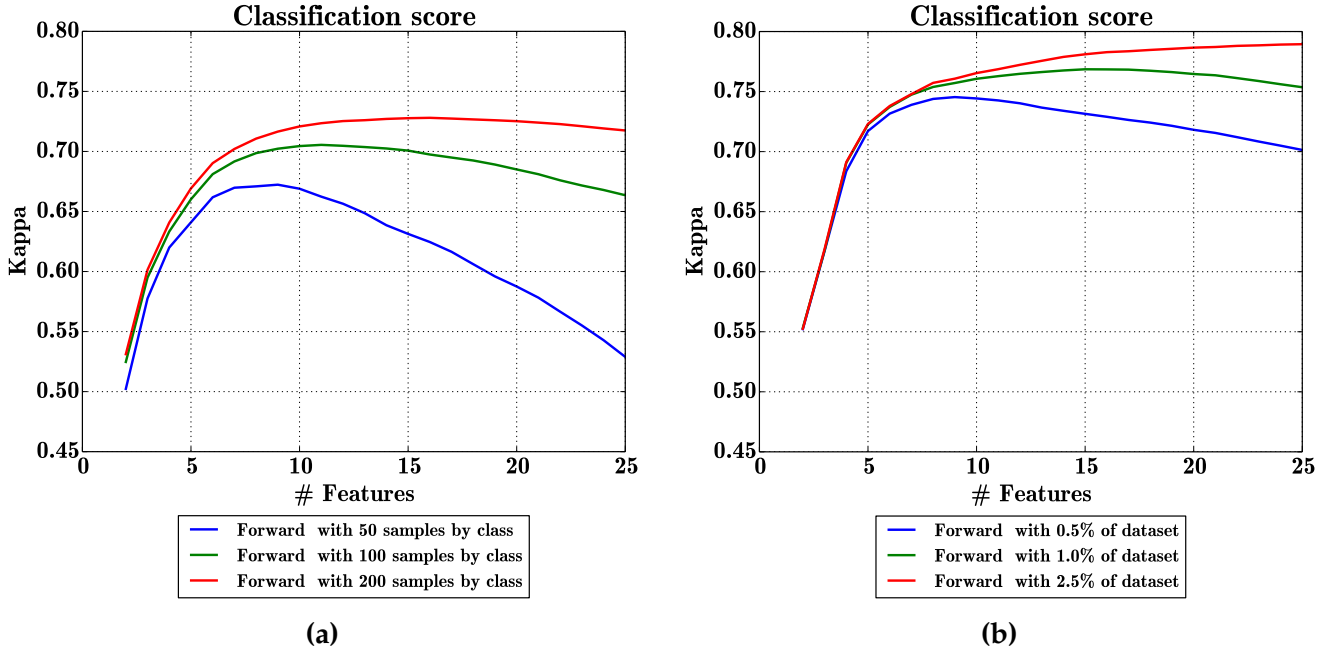


Figure 8: Classification results averaged on 20 trials using forward selection and Jeffries-Matusita distance as criterion: **(a)** with 50, 100, 200 samples by class in training set and **(b)** with 0.5%, 1%, 2.5% of dataset in training set keeping proportion.

that with this particular dataset, the SFFS does not produce better results than the other method. One should be careful not to draw conclusion too fast. It does not mean that the SFFS method has no advantages but rather that the SFFS advantages are highly related to the dataset structure and thus it is always good to check if the basic forward method is sufficient for a given dataset.

Now, if we take a closer look to the influence of criterion function, we can see from Figure 10 the results of classification for each of the criterion functions presented in Section 2.3.1. What we expected was that the criterion based on good classification mesure would be better because it would optimize directly the classification rate and maybe more flexible regarding to the Gaussian

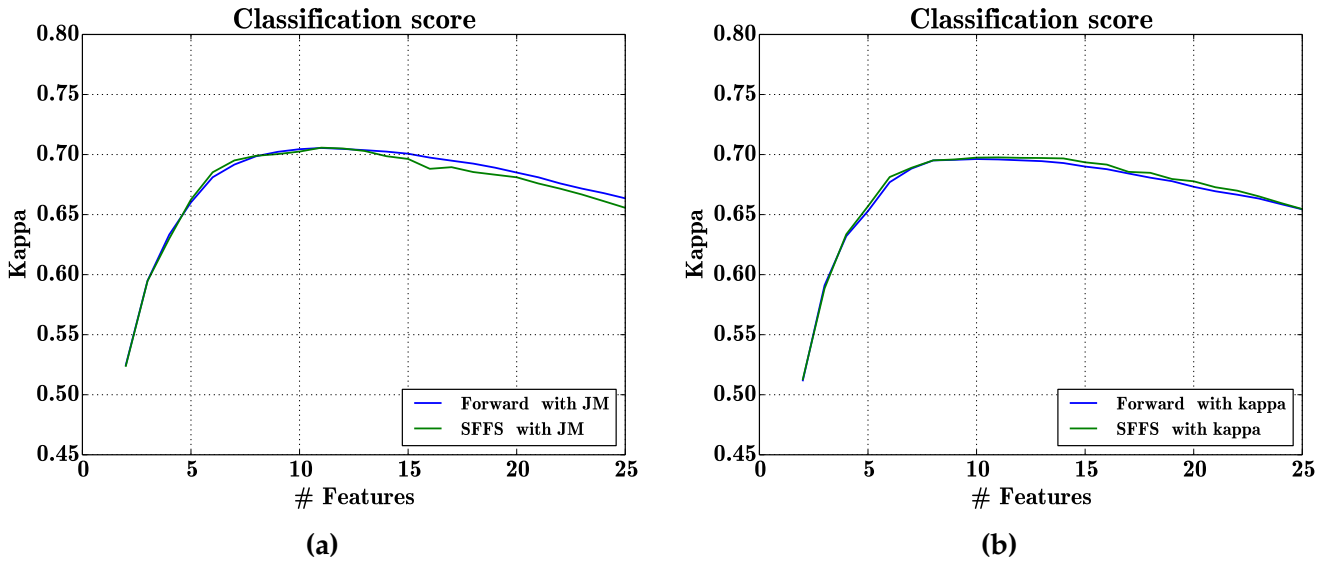


Figure 9: Classification results with 20 trials : **(a)** using Jeffries-Matusita distance as criterion and **(b)** using Cohen's kappa as criterion.

assumption. But it appears that for this given dataset, as good results are obtained with Jeffries-Matusita distance.

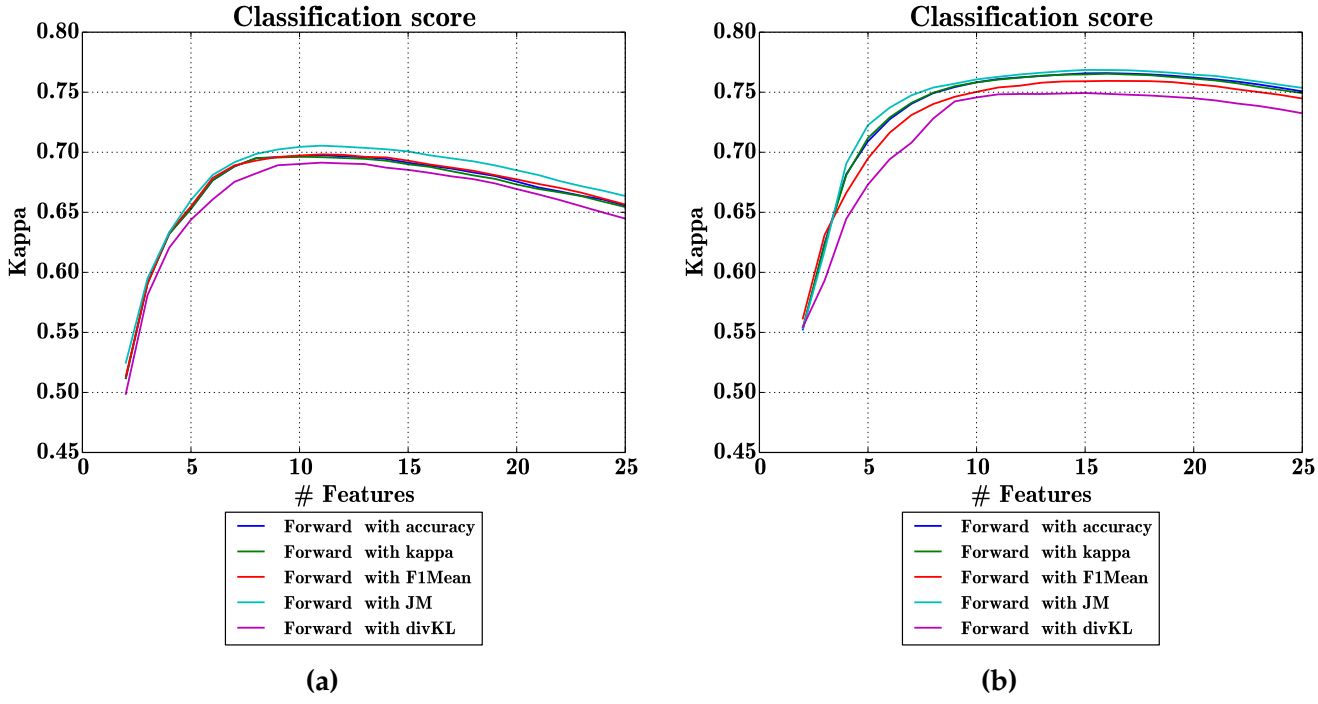


Figure 10: Classification results with different criterion function with 20 trials : **(a)** with a training composed of 100 samples for each class and **(b)** with a training set keeping class proportions composed of 1% of the dataset.

Finally, if we compare the various method and criterion function in term of computational time, we first notice that the SFFS method takes more time than the forward method which is actually a fact supported by the theoretical ground. It can also be noticed than the processing time for

selection do not increase with the number of samples in training set for criterion based on divergence (Kullback–Leibler divergence, Jeffries-Matusita distance) whereas, in the case of good classification criteria, the processing time rapidly increases. All processing time are presented in Figure 11.

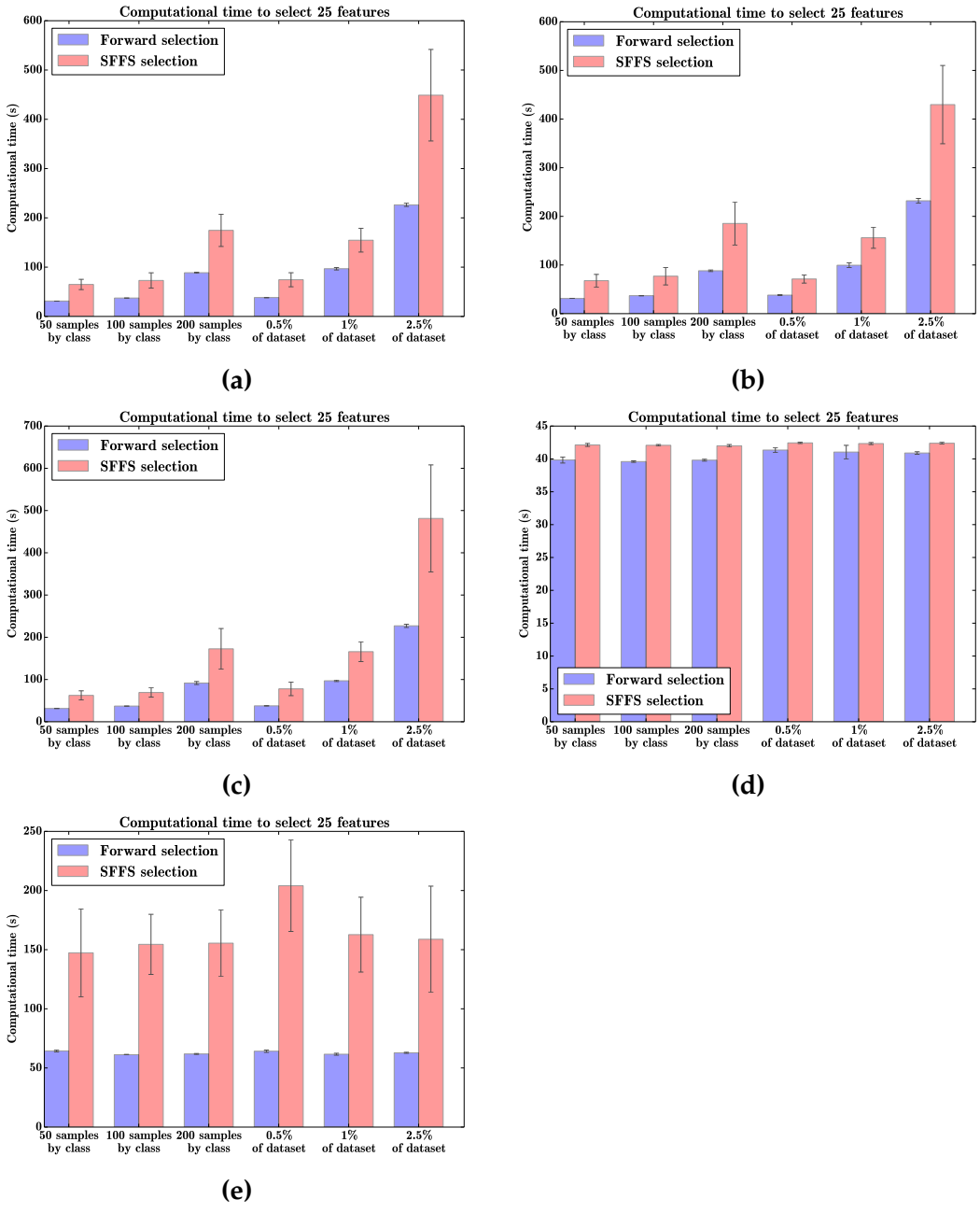


Figure 11: Processing time for selection with 20 trials : **(a)** using Cohen's kappa, **(b)** using overall accuracy, **(c)** using mean of F1-scores, **(d)** using KL divergence, **(e)** using JM distance.

4.2. Test: comparison to classical classifiers

4.2.1. Aisa dataset

We use the Aisa dataset described in Section 4.1.1 to compare performances of our classifiers to standard classifiers. The tests are conducted using OTB commandline applications.

We test 4 variations of our algorithm: one with ridge regularization and selection of the regularization constant (GMM ridge), one with forward selection and JM distance as criterion (GMM SFS JM), one with forward selection and Cohen's kappa as criterion (GMM SFS kappa) and one with floating forward selection and JM distance as criterion (GMM SFFS JM). We compare these classifiers to 3 OTB classifiers: the k-nearest-neighbors classifiers (KNN) with OTB default parameters, the Random Forest classifier with parameters optimized by gridsearch (200 trees, 40 max depth, 50 size of subset of variables) and the GMM classifier of OpenCV.

When creating training and validation sets, we take special care to assure that training samples are picked in distinct areas than test samples. In order to do it, we split the polygons of the shapefile in smaller polygons and then we take randomly 50% of the polygons for training and 50% for validation. Moreover 20 trials are run each time with a different training set (different polygons). An example of training and validation set is available in Appendix C.1. Additionally, it is important to understand that we do not use the whole training set for training our model but a given number of samples for each class. Table 3 presents the results of the experiment with mean and standard deviation over the 20 trials and Table 4 the corresponding processing time.

	Cohen's kappa		
# samples by class	250	500	1000
GMM SFS kappa	0.684 (0.025)	0.698 (0.027)	0.711 (0.028)
GMM SFS JM	0.680 (0.028)	0.700 (0.028)	0.710 (0.030)
GMM SFFS JM	0.680 (0.028)	0.700 (0.028)	0.710 (0.030)
GMM ridge	0.611 (0.040)	0.620 (0.036)	0.642 (0.034)
KNN	0.551 (0.035)	0.563 (0.033)	0.574 (0.030)
Random Forest	0.645 (0.026)	0.673 (0.023)	0.693 (0.023)

Table 3: Results of classification with sampling on separate polygons and 20 trials (standard deviation in parenthesis).

	Training time (s)			**Classification time (s)**		
# samples by class	250	500	1000	250	500	1000
GMM SFS kappa	257	496	955	7.7	8.6	8.7
GMM SFS JM	8.6	8.9	9.1	9.7	9.8	9.6
GMM SFFS JM	8.8	9.0	9.3	9.7	9.8	9.8
GMM ridge	71.7	105	167	530	530	530
KNN	8.9	19.6	59.7	387	639	887
Random Forest	24.5	49.3	105	33.0	41.7	45.9

Table 4: Mean processing time for training and classification for results of Table 3.

The results shows that, on this dataset, GMM classifiers with selection get the best classification rate followed by Random Forest. Especially with very few training samples, our algorithm seems to outperform the others. With more samples, Random Forest becomes more competitive with GMM classifiers.

In term of computational time, the GMM classifiers are as we expected very fast for classification and also for training if the criterion function is not a classification rate. The computation of the kappa seems to suffer from a lack of parallelization which could be improved.

As we expected, the OpenCv GMM classifier was not able to handle the high dimensionality and simply classify all samples in the same class resulting in a kappa of 0. Nevertheless, to compare with it, the dimension was reduced from 252 bands to 100 bands with a PCA. Table 5 and 6 presents the results obtained with this reduced dataset.

Three algorithms are tested with this reduced set: the GMM classifier of OpenCV library (GMM OpenCV), the GMM classifier we developed (GMM) and a GMM classifier with a sequential forward selection and Jeffries-Matusita distance as criterion (GMM SFS JM). As expected, there is no difference between our GMM classifier and OpenCV GMM classifier except that OpenCV classifier is faster for classification. We can also noticed that the GMM classifier with selection gets better results and classify faster for a very small cost of time during training.

	Cohen's kappa		
# samples by class	250	500	1000
GMM OpenCV	0.531 (0.034)	0.635 (0.024)	0.680 (0.025)
GMM	0.530 (0.034)	0.634 (0.024)	0.680 (0.025)
GMM SFS JM	0.692 (0.028)	0.704 (0.030)	0.711 (0.032)

Table 5: Results of classification with a reduced set of 100 features obtained by PCA and with sampling on separate polygons and 20 trials (standard deviation in parenthesis).

	Training time (s)			**Classification time (s)**		
# samples by class	250	500	1000	250	500	1000
GMM OpenCV	5.6	5.9	6.5	34.3	34.2	34.2
GMM	5.3	5.9	7.0	58.4	58.5	58.5
GMM SFS JM	6.3	6.3	6.6	7.7	7.7	7.7

Table 6: Mean processing time for training and classification for results of Table 5.

Classification images of the first trial are available in Appendix C.1.

4.2.2. Potsdam dataset

This new dataset is built from a dataset of remote sensing images distributed by the International Society for Photogrammetry and Remote Sensing (ISPRS)[6]. The dataset is composed of aerial images of the urban area of Potsdam. The area is cut into 38 patches of 6000x6000 pixels with a resolution of 5cm by pixel and 4 channels are available: Red, Blue, Green and Infrared (RGBIR). A Digital Surface Model with same resolution is also provided. The groundtruth for 24 tiles are provided with 6 classes: Low vegetation, High vegetation, Impervious surfaces, Buildings, Cars, Clutter. Figure 12 shows the complete area and Figure 13 a particular patch.

Class	Number of samples in 5_11	Number of samples in 5_12
Clutter		
Trees		
Cars		
Buildings		
Low vegetation		
Impervious surfaces		

Table 7: table
Repartition of classes in Aisa dataset.

In our experiment, we compute the following features using the RGBIR images:

[6] http://www2.isprs.org/commissions/comm3/wg4/2d-sem-label-potsdam.html

Figure 12: Full Potsdam dataset.

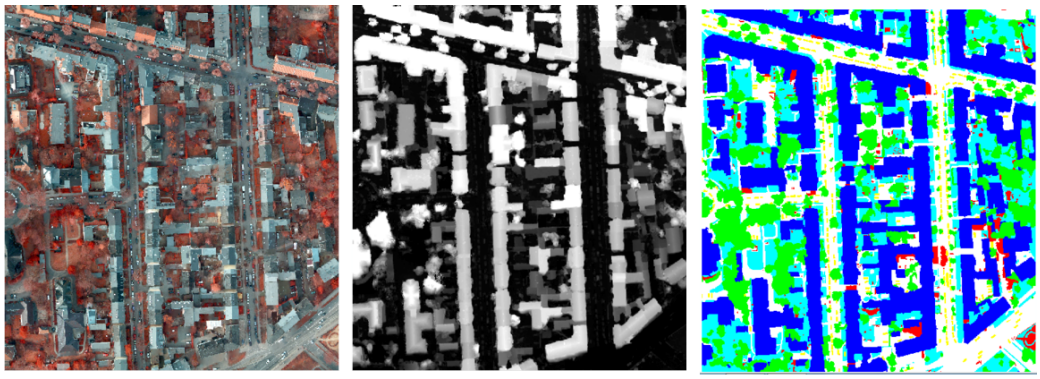


Figure 13: Example of patch with **(a)** orthophoto, **(b)** DSM and **(c)** groundtruth.

- 15 radiometric indexes: NDVI, TNDVI, RVI, SAVI, TSAVI, MSAVI, MSAVI2, GEMI, IPVI, NDWI2, NDTI, RI, CI, BI, BI2 (see [7]);
- morphological profile with reconstruction of each band with a disk of radius 5, 9, 13, 17, 21, 25, 29, 33, 37 and 41 (80 features)[23];
- attribute profile of each band with area as attribute and 1000, 2000, 5000, 10000 and 15000 as thresholds (40 features)[23];
- attribute profile of each band with diagonal of bounding box as attribute and 100, 200, 500, 1000 and 20000 as thresholds (40 features)[23];
- textural features with neighborhood of 19x19 pixels: mean, standard deviation, range and entropy (16 features)[23].

The DSM and the raw RGBIR image are added to these 191 features and then then stacked to create

[7] https://www.orfeo-toolbox.org//Applications/RadiometricIndices.html

a new image with 196 bands. The training is made with tile 5_11 and we test with tile 5_12. Table 8 presents the obtained results.

	Cohen's kappa	Training time (s)	Classification time (s)	# features used for classific
GMM SFS kappa	0.679 (0.007)	100	300	13.8
GMM SFS JM	0.453 (0.226)	1	400	29.4
GMM SFFS JM	0.453 (0.226)	1	400	29.4
GMM ridge	0.478 (0.014)	10	2000	all
KNN	0.468 (0.002)	1	1000	all
Random Forest	0.680 (0.005)	20	800	all

Table 8: Results of classification with 1000 samples by class and 5 trials (standard deviation in parenthesis).

As we expected, the results obtained with this second dataset does not support the same classifiers. The classification rate of GMM classifiers with Jeffries-Matusita distance as criterion function drops clearly and has a high standard deviation. It could be due to the fact that the Gaussian hypothesis, on which this metric relies a lot, is not enough verified. When the classification rate is directly used as criterion, the classifier manages to select the relevant features. It has to be noticed that with kappa only 10 to 15 features are selected when with Jeffries-Matusita distance, the maximum number of features is selected (set to 30).

The KNN classifier and the GMM classifier with ridge regularization are also outperformed even if they get stable results.

Finally, the Random Forest classifier and the GMM with kappa as criterion are from far the best classifiers. The only difference is in the processing time. Random Forest is faster for the training process when the GMM classifier is faster for classifying.

5. Conclusion

An algorithm for the classification of high dimension data as hyperspectral image has been presented. The classifier uses GMM model to select iteratively the most relevant features. Several variation of the algorithms has been explored. Experimentations show that allowing backward step in the selection to discard already selected features do not give significant advantage. Additionally, among all the measures tested to rank the features the Jeffries-Matusita distance seems the most accurate and the fastest when data distribution is almost Gaussian but if it is not the case, it seems preferable to use directly kappa as criterion function even if it implies a loss of time. We finally show that the developed GMM classifier performs at least as best as standard classifiers without features selection in particular Random Forest.

The second achievement of our work is the development of an efficient implementation using GMM properties to derive fast update rules of the model. The resulting code is available as a remote module of the Orfeo toolbox on Github and make it possible to process huge quantity of high dimension data.

More investigation is needed to confirm the limitation of the divergence measures when data distribution is not Gaussian. Moreover, the method should also be compared to other feature selection methods. Finally, an improvement could be made to increase the stability of the selected features. For example, with hyperspectral data, a selection of continuous intervals and not band is a possible solution which has already been explored in [24].

A. Hyperspectral imaging

Contrary to conventional color images which only have a red, green and blue components, hyperspectral images aims to capture information from across a much larger part of the electromagnetic spectrum. The spectrum is divided in smaller bands of frequencies chosen to cover the most interesting part of the spectrum relatively to the mission. In practice, hyperspectral images are composed of hundreds of bands. For example, the Aisa dataset presented in Section 4.1 is made of 252 bands which means 252 values to described a single pixel. An example of what can be an hyperspectral image is displayed in Figure 14.

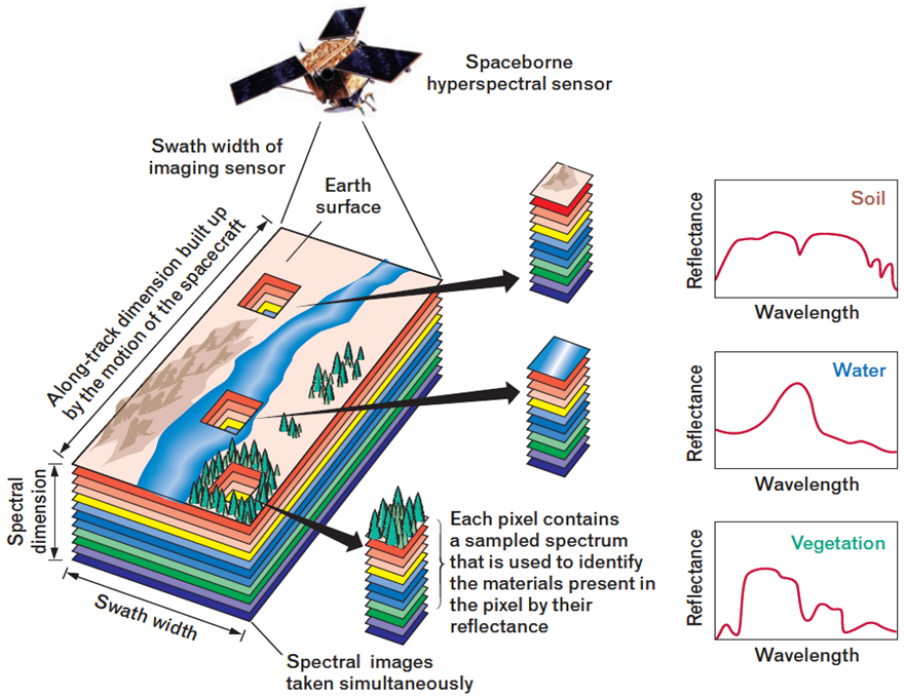


Figure 14: Example of hyperspectral image (Image from *G. Shaw et al.*).

The advantages of such imaging is to enable the identification of the different materials from their unique spectral responses. Such a process can be useful in numerous applications as agriculture management, oil prospection, pollution control, etc.

Meanwhile, these images possess several drawbacks. First, their acquisition requires specific sensors which are costly and bulky. Secondly, due to the tightness of a specific band, the energy of signal to measure is very small and thus limits the spacial resolution of the image (usually several tens of meters). And finally, specific algorithms need to be designed in order to handle the dimensionality of the data.

B. Update for cross validation (calculation)

B.1. Mean update

$$\begin{aligned}
\boldsymbol{\mu}_c^{n_c} &= \frac{1}{n_c}\sum_{j=1}^{n_c}\mathbf{x}_j \\
&= \frac{1}{n_c}\sum_{j=1}^{n_c-\nu_c}\mathbf{x}_j + \frac{1}{n_c}\sum_{j=n_c-\nu_c+1}^{n_c}\mathbf{x}_j \\
&= \frac{n_c-\nu_c}{n_c}\boldsymbol{\mu}_c^{n_c-\nu_c} + \frac{\nu_c}{n_c}\boldsymbol{\mu}_c^{\nu_c} \\
&= \boldsymbol{\mu}_c^{n_c-\nu_c} + \frac{\nu_c}{n_c}(\boldsymbol{\mu}_c^{\nu_c} - \boldsymbol{\mu}_c^{n_c-\nu_c})
\end{aligned}$$

$$\boxed{\boldsymbol{\mu}_c^{n_c} = \boldsymbol{\mu}_c^{n_c-\nu_c} + \tfrac{\nu_c}{n_c}(\boldsymbol{\mu}_c^{\nu_c} - \boldsymbol{\mu}_c^{n_c-\nu_c})} \tag{28}$$

$$\boxed{\boldsymbol{\mu}_c^{n_c-\nu_c} = \tfrac{n_c\boldsymbol{\mu}_c^{n_c} - \nu_c\boldsymbol{\mu}_c^{\nu_c}}{n_c-\nu_c}} \tag{29}$$

B.2. Covariance matrix update

$$\begin{aligned}
\boldsymbol{\Sigma}_c^{n_c} &= \frac{1}{n_c-1}\sum_{j=1}^{n_c}(\mathbf{x}_j - \boldsymbol{\mu}_c^{n_c})(\mathbf{x}_j - \boldsymbol{\mu}_c^{n_c})^t \\
&= \frac{1}{n_c-1}\sum_{j=1}^{n_c}(\mathbf{x}_j - \boldsymbol{\mu}_c^{n_c-\nu_c} - \frac{\nu_c}{n_c}(\boldsymbol{\mu}_c^{\nu_c} - \boldsymbol{\mu}_c^{n_c-\nu_c}))(\mathbf{x}_j - \boldsymbol{\mu}_c^{n_c-\nu_c} - \frac{\nu_c}{n_c}(\boldsymbol{\mu}_c^{\nu_c} - \boldsymbol{\mu}_c^{n_c-\nu_c}))^t \\
&= \frac{1}{n_c-1}\sum_{j=1}^{n_c}(\mathbf{x}_j - \boldsymbol{\mu}_c^{n_c-\nu_c})(\mathbf{x}_j - \boldsymbol{\mu}_c^{n_c-\nu_c})^t \\
&\quad + \frac{\nu_c^2}{n_c^2}(\boldsymbol{\mu}_c^{\nu_c} - \boldsymbol{\mu}_c^{n_c-\nu_c})(\boldsymbol{\mu}_c^{\nu_c} - \boldsymbol{\mu}_c^{n_c-\nu_c})^t \\
&\quad - \frac{\nu_c}{n_c}(\mathbf{x}_j - \boldsymbol{\mu}_c^{n_c-\nu_c})(\boldsymbol{\mu}_c^{\nu_c} - \boldsymbol{\mu}_c^{n_c-\nu_c})^t \\
&\quad - \frac{\nu_c}{n_c}(\boldsymbol{\mu}_c^{\nu_c} - \boldsymbol{\mu}_c^{n_c-\nu_c})(\mathbf{x}_j - \boldsymbol{\mu}_c^{n_c-\nu_c})^t
\end{aligned}$$

- $(\boldsymbol{\mu}_c^{\nu_c} - \boldsymbol{\mu}_c^{n_c-\nu_c})(\boldsymbol{\mu}_c^{\nu_c} - \boldsymbol{\mu}_c^{n_c-\nu_c})^t = \frac{n_c^2}{(n_c-\nu_c)^2}(\boldsymbol{\mu}_c^{\nu_c} - \boldsymbol{\mu}_c^{n_c})(\boldsymbol{\mu}_c^{\nu_c} - \boldsymbol{\mu}_c^{n_c})^t$
- $\frac{1}{n_c}\sum_{j=1}^{n_c}(\mathbf{x}_j - \boldsymbol{\mu}_c^{n_c-\nu_c})(\boldsymbol{\mu}_c^{\nu_c} - \boldsymbol{\mu}_c^{n_c-\nu_c})^t = \frac{n_c\nu_c}{(n_c-\nu_c)^2}(\boldsymbol{\mu}_c^{\nu_c} - \boldsymbol{\mu}_c^{n_c})(\boldsymbol{\mu}_c^{\nu_c} - \boldsymbol{\mu}_c^{n_c})^t$
- $\frac{1}{n_c}\sum_{j=1}^{n_c}(\boldsymbol{\mu}_c^{\nu_c} - \boldsymbol{\mu}_c^{n_c-\nu_c})(\mathbf{x}_j - \boldsymbol{\mu}_c^{n_c-\nu_c})^t = \frac{n_c\nu_c}{(n_c-\nu_c)^2}(\boldsymbol{\mu}_c^{\nu_c} - \boldsymbol{\mu}_c^{n_c})(\boldsymbol{\mu}_c^{\nu_c} - \boldsymbol{\mu}_c^{n_c})^t$

$$
\begin{aligned}
\boldsymbol{\Sigma}_c^{n_c} &= \frac{1}{n_c-1}\sum_{j=1}^{n_c}(\mathbf{x}_j-\boldsymbol{\mu}_c^{n_c-\nu_c})(\mathbf{x}_j-\boldsymbol{\mu}_c^{n_c-\nu_c})^t - \frac{\nu_c^2}{(n_c-\nu_c)^2}(\boldsymbol{\mu}_c^{\nu_c}-\boldsymbol{\mu}_c^{n_c})(\boldsymbol{\mu}_c^{\nu_c}-\boldsymbol{\mu}_c^{n_c})^t \\
&= \frac{1}{n_c-1}\sum_{j=1}^{n_c-\nu_c}(\mathbf{x}_j-\boldsymbol{\mu}_c^{n_c-\nu_c})(\mathbf{x}_j-\boldsymbol{\mu}_c^{n_c-\nu_c})^t + \frac{1}{n_c-1}\sum_{j=n_c-\nu_c+1}^{n_c}(\mathbf{x}_j-\boldsymbol{\mu}_c^{n_c-\nu_c})(\mathbf{x}_j-\boldsymbol{\mu}_c^{n_c-\nu_c})^t \\
&\quad - \frac{n_c\nu_c^2}{(n_c-1)(n_c-\nu_c)^2}(\boldsymbol{\mu}_c^{\nu_c}-\boldsymbol{\mu}_c^{n_c})(\boldsymbol{\mu}_c^{\nu_c}-\boldsymbol{\mu}_c^{n_c})^t \\
&= \frac{n_c-\nu_c-1}{n_c-1}\boldsymbol{\Sigma}_c^{n_c-\nu_c} + \frac{1}{n_c-1}\sum_{j=n_c-\nu_c+1}^{n_c}(\mathbf{x}_j-\boldsymbol{\mu}_c^{n_c-\nu_c})(\mathbf{x}_j-\boldsymbol{\mu}_c^{n_c-\nu_c})^t \\
&\quad - \frac{n_c\nu_c^2}{(n_c-1)(n_c-\nu_c)^2}(\boldsymbol{\mu}_c^{\nu_c}-\boldsymbol{\mu}_c^{n_c})(\boldsymbol{\mu}_c^{\nu_c}-\boldsymbol{\mu}_c^{n_c})^t
\end{aligned}
$$

$$
\begin{aligned}
\sum_{j=n_c-\nu_c+1}^{n_c}(\mathbf{x}_j-\boldsymbol{\mu}_c^{n_c-\nu_c})(\mathbf{x}_j-\boldsymbol{\mu}_c^{n_c-\nu_c})^t &= \sum_{j=n_c-\nu_c+1}^{n_c}(\mathbf{x}_j-\boldsymbol{\mu}_c^{\nu_c}+\boldsymbol{\mu}_c^{\nu_c}-\boldsymbol{\mu}_c^{n_c-\nu_c})(\mathbf{x}_j-\boldsymbol{\mu}_c^{\nu_c}+\boldsymbol{\mu}_c^{\nu_c}-\boldsymbol{\mu}_c^{n_c-\nu_c})^t \\
&= (\nu_c-1)\boldsymbol{\Sigma}_c^{\nu_c} + \nu_c(\boldsymbol{\mu}_c^{\nu_c}-\boldsymbol{\mu}_c^{n_c-\nu_c})(\boldsymbol{\mu}_c^{\nu_c}-\boldsymbol{\mu}_c^{n_c-\nu_c})^t \\
&\quad + \sum_{j=n_c-\nu_c+1}^{n_c}(\mathbf{x}_j-\boldsymbol{\mu}_c^{\nu_c})(\boldsymbol{\mu}_c^{\nu_c}-\boldsymbol{\mu}_c^{n_c-\nu_c})^t \\
&\quad + \sum_{j=n_c-\nu_c+1}^{n_c}(\boldsymbol{\mu}_c^{\nu_c}-\boldsymbol{\mu}_c^{n_c-\nu_c})(\mathbf{x}_j-\boldsymbol{\mu}_c^{\nu_c})^t \\
&= (\nu_c-1)\boldsymbol{\Sigma}_c^{\nu_c} + \frac{n_c^2\nu_c}{(n_c-\nu_c)^2}(\boldsymbol{\mu}_c^{\nu_c}-\boldsymbol{\mu}_c^{n_c})(\boldsymbol{\mu}_c^{\nu_c}-\boldsymbol{\mu}_c^{n_c})^t
\end{aligned}
$$

$$
\boxed{\boldsymbol{\Sigma}_c^{n_c} = \tfrac{n_c-\nu_c-1}{n_c-1}\boldsymbol{\Sigma}_c^{n_c-\nu_c} + \tfrac{\nu_c-1}{n_c-1}\boldsymbol{\Sigma}_c^{\nu_c} + \tfrac{n_c\nu_c}{(n_c-1)(n_c-\nu_c)}(\boldsymbol{\mu}_c^{\nu_c}-\boldsymbol{\mu}_c^{n_c})(\boldsymbol{\mu}_c^{\nu_c}-\boldsymbol{\mu}_c^{n_c})^t} \tag{30}
$$

$$
\boxed{\boldsymbol{\Sigma}_c^{n_c-\nu_c} = \tfrac{1}{n_c-\nu_c-1}((n_c-1)\boldsymbol{\Sigma}_c^{n_c} - (\nu_c-1)\boldsymbol{\Sigma}_c^{\nu_c} - \tfrac{n_c\nu_c}{(n_c-\nu_c)}(\boldsymbol{\mu}_c^{\nu_c}-\boldsymbol{\mu}_c^{n_c})(\boldsymbol{\mu}_c^{\nu_c}-\boldsymbol{\mu}_c^{n_c})^t\ )} \tag{31}
$$

C. Classification results

C.1. Aisa dataset classification

References

[1] C. Bouveyron and C. Brunet-Saumard, "Model-based clustering of high-dimensional data: A review," *Computational Statistics & Data Analysis*, vol. 71, pp. 52–78, 2014.

[2] E. Christophe, J. Michel, and J. Inglada, "Remote sensing processing: From multicore to gpu," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 4, no. 3, pp. 643–652, 2011.

[3] A. Plaza, Q. Du, Y.-L. Chang, and R. L. King, "High performance computing for hyperspectral remote sensing," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 4, no. 3, pp. 528–544, 2011.

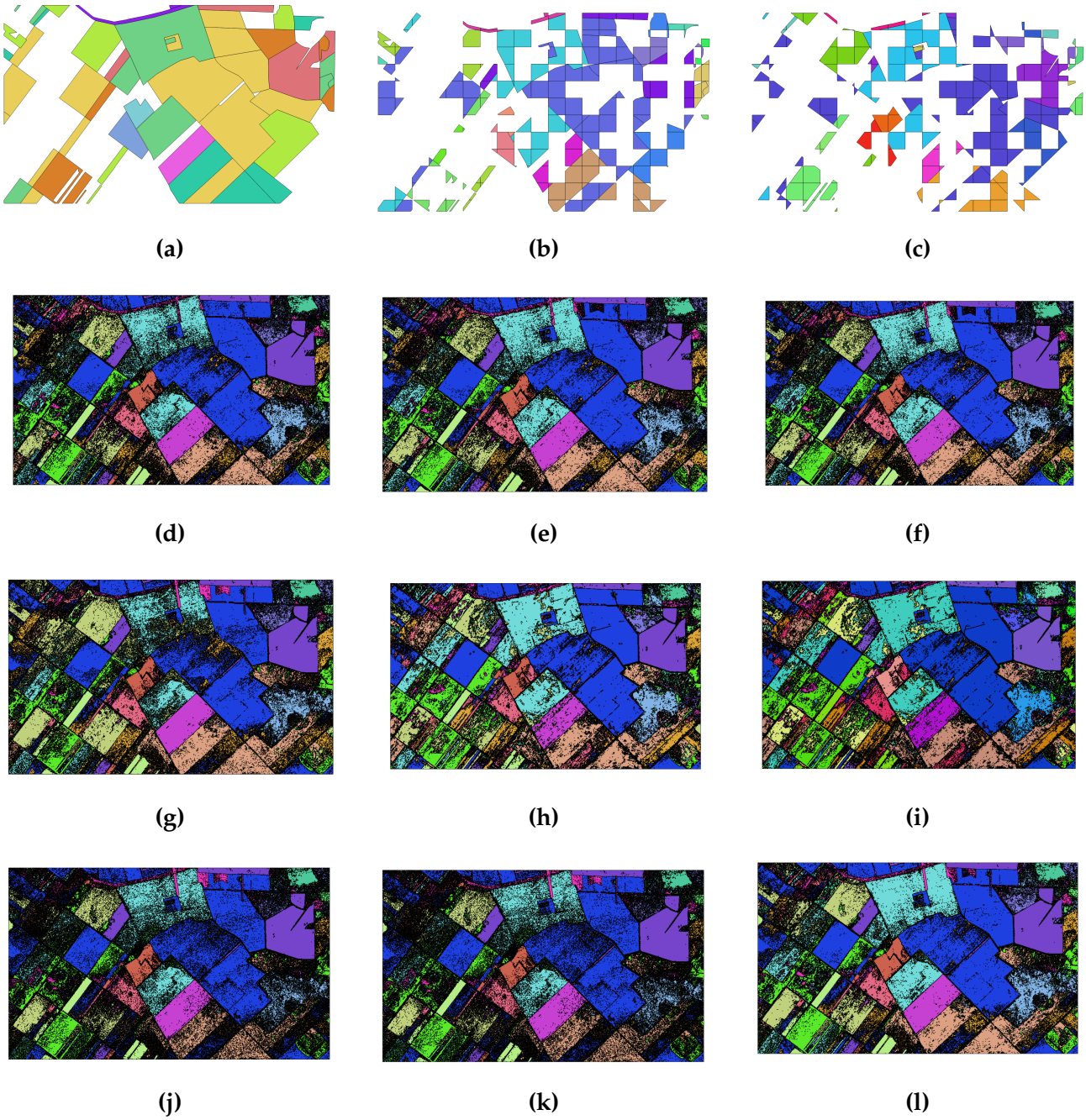


Figure 15: Aisa classification results: **(a)** groundtruth, **(b)** training polygons, **(c)** test polygons, **(d)** GMM SFS kappa, **(e)** GMM SFS JM, **(f)** GMM SFFS JM, **(g)** GMM ridge, **(h)** KNN, **(i)** Random Forest, **(j)** GMM OpenCV (reduced features), **(k)** GMM (reduced features), **(l)** GMM SFS JM (reduced features).

[4] L. O. Jimenez and D. A. Landgrebe, "Supervised classification in high-dimensional space: geometrical, statistical, and asymptotical properties of multivariate data," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 28, no. 1, pp. 39–54, 1998.

[5] F. E. Fassnacht, C. Neumann, M. Förster, H. Buddenbaum, A. Ghosh, A. Clasen, P. K. Joshi, and B. Koch, "Comparison of feature reduction algorithms for classifying tree species with hyperspectral data on three central european test sites," *IEEE Journal of Selected Topics in Applied*

Earth Observations and Remote Sensing, vol. 7, no. 6, pp. 2547–2561, 2014.

[6] I. Guyon, S. Gunn, M. Nikravesh, and L. A. Zadeh, *Feature Extraction: Foundations and Applications (Studies in Fuzziness and Soft Computing)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[7] M. Fauvel, C. Dechesne, A. Zullo, and F. Ferraty, "Fast forward feature selection of hyperspectral images for classification with gaussian mixture models," *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of*, vol. 8, no. 6, pp. 2824–2831, 2015.

[8] E. Christophe, J. Inglada, and A. Giros, "Orfeo toolbox: a complete solution for mapping from high resolution satellite images," *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 37, pp. 1263–1268, 2008.

[9] R. G. Congalton and K. Green, *Assessing the accuracy of remotely sensed data: principles PCAd practices*. CRC press, 2008.

[10] T. J. Hastie, R. J. Tibshirani, and J. H. Friedman, *The elements of statistical learning : data mining, inference, and prediction*. Springer series in statistics, Springer, 2009.

[11] L. Bruzzone and C. Persello, "A novel approach to the selection of spatially invariant features for the classification of hyperspectral images with improved generalization capability," *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 47, no. 9, pp. 3180–3191, 2009.

[12] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970.

[13] L. Bruzzone, F. Roli, and S. B. Serpico, "An extension of the jeffreys-matusita distance to multiclass cases for feature selection," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 33, no. 6, pp. 1318–1321, 1995.

[14] J. Biesiada and W. Duch, "Feature selection for high-dimensional data—a pearson redundancy based filter," in *Computer Recognition Systems 2*, pp. 242–249, Springer, 2007.

[15] B. Demir and S. Ertürk, "Phase correlation based redundancy removal in feature weighting band selection for hyperspectral images," *International journal of Remote sensing*, vol. 29, no. 6, pp. 1801–1807, 2008.

[16] P. M. Narendra and K. Fukunaga, "A branch and bound algorithm for feature subset selection," *IEEE Transactions on Computers*, vol. 100, no. 9, pp. 917–922, 1977.

[17] A. W. Whitney, "A direct method of nonparametric measurement selection," *IEEE Transactions on Computers*, vol. 100, no. 9, pp. 1100–1103, 1971.

[18] P. Somol, P. Pudil, J. Novovičová, and P. Paclık, "Adaptive floating search methods in feature selection," *Pattern recognition letters*, vol. 20, no. 11, pp. 1157–1163, 1999.

[19] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, "Gene selection for cancer classification using support vector machines," *Machine learning*, vol. 46, no. 1-3, pp. 389–422, 2002.

[20] J. Weston, A. Elisseeff, B. Schölkopf, and M. Tipping, "Use of the zero-norm with linear models and kernel methods," *Journal of machine learning research*, vol. 3, no. Mar, pp. 1439–1461, 2003.

[21] A. R. Webb, *Statistical pattern recognition*. John Wiley & Sons, 2003.

[22] G. Hughes, "On the mean accuracy of statistical pattern recognizers," *IEEE transactions on information theory*, vol. 14, no. 1, pp. 55–63, 1968.

[23] D. Tuia, R. Flamary, and N. Courty, "Multiclass feature learning for hyperspectral image classification: Sparse and hierarchical solutions," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 105, pp. 272–285, 2015.

[24] S. B. Serpico and G. Moser, "Extraction of spectral channels from hyperspectral images for classification purposes," *IEEE transactions on geoscience and remote sensing*, vol. 45, no. 2, pp. 484–495, 2007.