Report : Operational Feature Selection in Gaussian Mixture Models

Adrien Lagrange
Dynafor

July 21, 2016

1. Introduction

With the increasing number of remote sensing missions, the quantity of data available for a given landscape becomes larger and larger. In particular, the European satellite Sentinel-2 was launched successfully recently and the hyperspectral and hypertemporal data produced by this mission will be fully available at the end of 2016[1]. Therefore, data is more and more difficult to process because computational and also statistical limits. Hence, in many remote sensing applications, the extraction of features from a large amount of available data is required.

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 as explained in [1]. 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.

There are two ways to reduce dimension: features extraction and feature selection. Feature extraction means creating new features in combining the existing ones, for example linear combination as in Principal Component Analysis [2]. To the contrary, features selection select a subset of the existing features which 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 method for feature selection. However, they usually do not scale well with the number of pixels to be processed as shown in [3]. Nevertheless, methods based on Gaussian Mixture Models (GMM) have several interesting properties described in [4] 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.

The objectives of the internship is to develop and implement a feature selection method based on GMM. The remaining of this mid-term report is organized as follows.

2. Non linear parsimonious features selection

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.

[1] https://sentinel2.cnes.fr/en/sentinel-2-0

2.1. Gaussian Mixture Models

The hypothesis of Gaussian mixture models (GMM) is that the distribution of 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. The Gaussian model assumes that each f_c is , conditionally to c, a Gaussian distribution of parameters $\boldsymbol{\mu}_c$ and $\boldsymbol{\Sigma}_c$ ($f_c(\mathbf{x}|\theta) = \mathcal{N}_c(\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$) 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 following 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 using the Bayes' law, the maximum a posteriori (MAP) estimate is used and thus the decision rule can be written as

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

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 the covariance and its inverse 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 samples are labeled available because of the difficulty and the cost to collect ground-truth.

A lack of samples induces ill-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 method in order to reduce the dimensionality of the samples.

In our study based on GMM, a Ridge regularization method described in Section 3 is implemented but we mainly focus on a feature selection method named sequential forward features selection presented in Section 4.1 and an amelioration of this algorithm the sequential floating forward features selection introduced in Section 4.2.

In order to evaluate the benefits of such methods for classification, we introduce in the next section several functions called criterion functions. These functions aim to evaluate either a criterion of good classification or the separability/similarity of class distributions.

2.2. Criterion function

2.2.1. Measures of good classification

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, True Negative (TN) corresponding to good classification of the other class, False Negative (FN) corresponding to the samples of the class labeled as an other class and the False Positive (FP) corresponding to the samples wrongly classified as part of of this class. Figure 1 illustrates the definition of this values.

		Prediction outcome	
		p	**n**
Actual value	**p′**	True Positive	False Negative
	n′	False Positive	True Negative

Figure 1: Confusion matrix

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 but is hard to interpret.

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).

$$\text{Overall Accuracy} = \frac{\text{TP}}{n} \tag{6}$$

$$\text{Kappa} = \frac{pa - pr}{1 - pr} \tag{7}$$

$$\text{F1 Mean} = \frac{2\text{TP}}{2\text{TP} + \text{FN} + \text{FP}} \tag{8}$$

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 compute these metrics, the predicted labels are needed and so we need to compute the decision function (Equation 5). Moreover to estimate this measures of good classification, a cross-validation method is used which reduces both bias and variance of the estimation.

2.2.2. Measures of similarity between distributions

The Kullback–Leibler divergence is a measure of distance between two distributions. It actually measures the amount of information lost when the first distribution is approximated by the second one. The formal definition is

$$\text{Div}_{KL}(c_i, c_j) = \int_{\mathbf{x}} p(\mathbf{x}|c_i) \ln(\frac{p(\mathbf{x}|c_i)}{p(\mathbf{x}|c_j)}) d\mathbf{x} \tag{9}$$

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

$$\text{Div}_{KL}(c_i, c_j) = \frac{1}{2} \left(\text{Tr}(\mathbf{\Sigma}_{c_i}^{-1} \mathbf{\Sigma}_{c_j}) + (\boldsymbol{\mu}_{c_i} - \boldsymbol{\mu}_{c_i})^t \mathbf{\Sigma}_{c_i}^{-1} (\boldsymbol{\mu}_{c_i} - \boldsymbol{\mu}_{c_i}) - d + \log \left(\frac{|\mathbf{\Sigma}_{c_i}|}{|\mathbf{\Sigma}_{c_j}|} \right) \right) \tag{10}$$

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

It can be noticed that the KL divergence is not symmetric 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

$$\begin{aligned} \text{SKL}_{ij} &= \text{Div}_{KL}(c_i, c_j) + \text{Div}_{KL}(c_j, c_i) \\ &= \frac{1}{2} \left(\text{Tr}(\mathbf{\Sigma}_{c_i}^{-1} \mathbf{\Sigma}_{c_j} + \mathbf{\Sigma}_{c_j}^{-1} \mathbf{\Sigma}_{c_i}) + (\boldsymbol{\mu}_{c_i} - \boldsymbol{\mu}_{c_i})^t (\mathbf{\Sigma}_{c_i}^{-1} + \mathbf{\Sigma}_{c_j}^{-1}) (\boldsymbol{\mu}_{c_i} - \boldsymbol{\mu}_{c_i}) - 2d \right) \end{aligned} \tag{11}$$

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

$$\text{C}_{SKL} = \sum_{i=1}^{C} \sum_{j=i+1}^{C} \pi_{c_i} \pi_{c_j} \text{SKL}_{ij} \tag{12}$$

The Bhattacharyya distance is an other measure of similarity between two distributions. In fact, we do not discuss the efficiency of this measure in our work and simply used it to compute the next measure. The Bhattacharyya distance is computed as follows

$$\text{B}_{ij} = -\ln \left(\int_{\mathbf{x}} \sqrt{p(\mathbf{x}|c_i) p(\mathbf{x}|c_j)} d\mathbf{x} \right) \tag{13}$$

And in the case of Gaussian model:

$$\text{B}_{ij} = \frac{1}{8} (\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)^t \left(\frac{\mathbf{\Sigma}_i + \mathbf{\Sigma}_j}{2} \right)^{-1} (\boldsymbol{\mu}_i - \boldsymbol{\mu}_j) + \frac{1}{2} \ln \left(\frac{|\mathbf{\Sigma}_i + \mathbf{\Sigma}_j|}{\sqrt{|\mathbf{\Sigma}_i||\mathbf{\Sigma}_j|}} \right) \tag{14}$$

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

$$\mathrm{JM}_{ij} = \sqrt{\int_{\mathbf{x}} \left[\sqrt{p(\mathbf{x}|c_i)} - \sqrt{p(\mathbf{x}|c_j)} \right]^2 d\mathbf{x}} \tag{15}$$

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

$$\mathrm{JM}_{ij} = \sqrt{2\{1 - \exp[-B_{ij}]\}} \tag{16}$$

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 [5], 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.

3. Ridge Regularization

3.1. Principe of 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 ill-conditioned matrix. 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 to each eigenvalues but it is not compulsory in a general case.

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

$$Q_c(\mathbf{x}) = -(\mathbf{x} - \boldsymbol{\mu}_c)^t \mathbf{Q}_c (\mathbf{\Lambda}_c + \tau\mathbf{I})^{-1} \mathbf{Q}_c^t (\mathbf{x} - \boldsymbol{\mu}_c) - \log((2\pi)^d |\mathbf{\Sigma}_c + \tau\mathbf{I}|) + 2\log(\pi_c) \tag{17}$$

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 element of the matrix.

3.2. Implementation

The main steps of the training algorithm when using Ridge regularization 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 and to do so, a gridsearch is performed with a set of values $\mathcal{P}$ given by the user. Moreover, to have a good estimation of the quality of the classification with a given τ a cross-validation method is used. 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.

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

Algorithm 1 Ridge Regularization training steps

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

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**
  Learn GMM with $\mathcal{S} \setminus \mathcal{S}_i$
  Compute decomposition of covariance matrices of each classes s.t. $\mathbf{\Sigma}_c = \mathbf{Q}_c \mathbf{\Lambda}_c \mathbf{Q}_c^t$
  for all $\tau \in \mathcal{P}$ **do**
    Estimate classification rate on $\mathcal{S}_i$ using Equation 17
  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^*
Learn GMM with the whole training set $\mathcal{S}$ and parametrize the GMM with τ^*

only once to test all τ. The covariance matrix is diagonalized before selecting τ and the eigenvalues and eigenvectors are stored and used to test all τ using Equation 17.

Finally, another trick could used to optimize the gridsearch. As explained in the next section, 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.

4. Features selection

We recall that the aim of feature selection method is to select a subset of variables to described each samples and so to get rid of high dimensionality and its curse. It is also important to underline that the proposed method is a *selection* method and not an *extraction* method. It means the subset of variables obtained at the end 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.

Two selection algorithms are presented the sequential forward features selection method (Section 4.1) and the sequential floating forward feature selection method (4.2) the second being a more complex variation of the first.

4.1. 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 2.2 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.

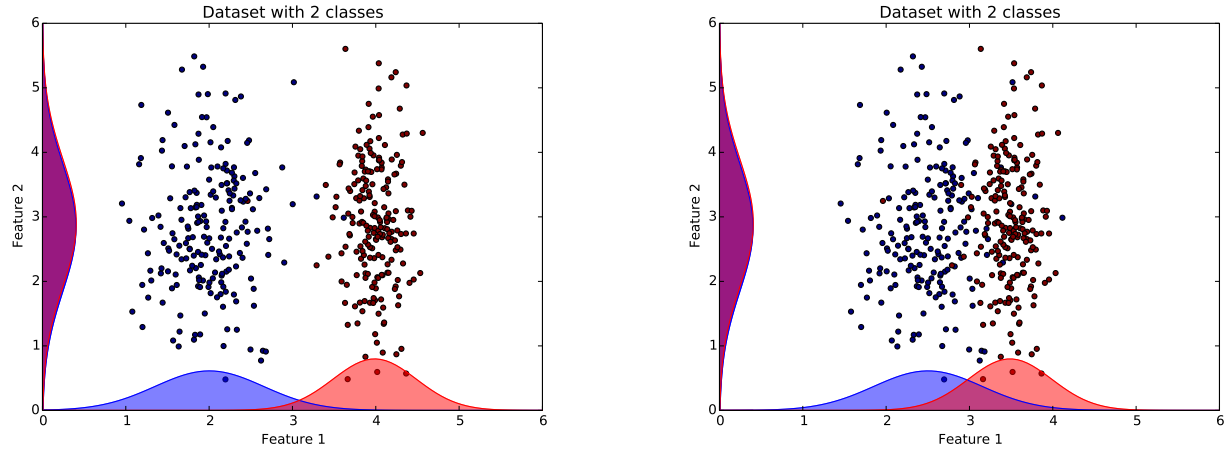
To summarize, the features are selected one by one and the set of selected features are never put into questions. It results in a reasonable computational time but a suboptimal solution to the selection problem.

Algorithm 2 Sequential forward features selection

Require: $\mathcal{S}, J, \text{maxVarNb}$

$S = \emptyset$
$F = \{\text{all variables}\}$
while $textcard(S) \leq maxVarNb$ **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 2 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 but it is not necessary in real case.



(a) **(b)**

Figure 2: 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.

In the case of the example presented in Figure 2, we compute the various criterion functions using a single feature and summarize it in Table 1. It corresponds to first step of the SFS. And 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 features to perform classification.

4.2. 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

	Case a		Case b	
	Feat 1	Feat 2	Feat 1	Feat 2
Overall Accuracy	0.995	0.445	0.935	0.445
Cohen's kappa	0.99	-0.11	0.87	-0.11
F1-score mean	0.995	0.43	0.935	0.43
KL divergence	13.04	0.25	3.48	0.25
Jeffrey-Matusita distance	0.35	0.19	0.31	0.19

Table 1: Criterion functions values corresponding to Figure 2 dataset (50% for training/50% for testing).

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.

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 of the complexity of the dataset.

4.3. Implementation

In term of computational efficiency, the main issue is to find an efficient way to compute the mean vector and the covariance matrix and its inverse and to do it only when it is absolutely necessary. More specifically, we develop an optimized way to derive the submodel for cross-validation from the full model and we minimize the number of costly operations during a step of the selection algorithm.

4.3.1. Update for cross validation

Based on [3], 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 A)

$$\boldsymbol{\mu}_c^{n_c-\nu_c} = \frac{n_c\boldsymbol{\mu}_c^{n_c} - \nu_c\boldsymbol{\mu}_c^{\nu_c}}{n_c-\nu_c} \tag{18}$$

$$\boldsymbol{\Sigma}_c^{n_c-\nu_c} = \frac{1}{n_c-\nu_c-1}((n_c-1)\boldsymbol{\Sigma}_c^{n_c} - (\nu_c-1)\boldsymbol{\Sigma}_c^{\nu_c} - \frac{n_c\nu_c}{(n_c-\nu_c)}(\boldsymbol{\mu}_c^{\nu_c}-\boldsymbol{\mu}_c^{n_c})(\boldsymbol{\mu}_c^{\nu_c}-\boldsymbol{\mu}_c^{n_c})^t) \tag{19}$$

where ν_c is the number of samples of class c removed from the set.

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.

4.3.2. 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.

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

We note $\mathbf{\Sigma}^{(k-1)}$ the covariance matrix of the $k-1^{th}$ iteration, i.e., the covariance of the selected features and $\mathbf{\Sigma}^{(k)}$ the covariance matrix of the k^{th} iteration, i.e., the covariance of the augmented set. Then, the inverse of the covariance matrix $(\mathbf{\Sigma}^{(k)})^{-1}$, the quadratical term $(\mathbf{x}^{(k)})^t(\mathbf{\Sigma}^{(k)})^{-1}\mathbf{x}^{(k)}$ and the determinant $\log|\mathbf{\Sigma}^{(k)}|$ 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 [4] (chapter 9.2).

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

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

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

$$\boxed{(\mathbf{\Sigma}^{(k)})^{-1} = \begin{bmatrix} (\mathbf{\Sigma}^{(k-1)})^{-1} + \frac{1}{\alpha}(\mathbf{\Sigma}^{(k-1)})^{-1}\mathbf{u}\mathbf{u}^t(\mathbf{\Sigma}^{(k-1)})^{-1} & -\frac{1}{\alpha}(\mathbf{\Sigma}^{(k-1)})^{-1}\mathbf{u} \\ -\frac{1}{\alpha}\mathbf{u}^t(\mathbf{\Sigma}^{(k-1)})^{-1} & \frac{1}{\alpha} \end{bmatrix}} \tag{20}$$

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

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 $(\mathbf{\Sigma}^{(k)})^{-1}$. Then, if we use the following notation for the previous matrix

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

with $\mathbf{A} = (\mathbf{\Sigma}^{(k-1)})^{-1} + \frac{1}{\alpha}(\mathbf{\Sigma}^{(k-1)})^{-1}\mathbf{u}\mathbf{u}^t(\mathbf{\Sigma}^{(k-1)})^{-1}$ and $\mathbf{v} = -\frac{1}{\alpha}(\mathbf{\Sigma}^{(k-1)})^{-1}\mathbf{u}$, we can calculate

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

$$\boxed{(\mathbf{\Sigma}^{(k-1)})^{-1} = \mathbf{A} - \alpha\mathbf{v}\mathbf{v}^t} \tag{21}$$

A formula can also be deduced for the quadratical term of the decision function. If we note

$$(\mathbf{x}^{(k)})^t = \begin{bmatrix} (\mathbf{x}^{(k-1)})^t & x^(k) \end{bmatrix},$$

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

$$\boxed{(\mathbf{x}^{(k)})^t(\boldsymbol{\Sigma}^{(k)})^{-1}\mathbf{x}^{(k)} = (\mathbf{x}^{(k-1)})^t(\boldsymbol{\Sigma}^{(k-1)})^{-1}\mathbf{x}^{(k-1)} + \alpha(\begin{bmatrix} \mathbf{v}^t & \frac{1}{\alpha} \end{bmatrix}\mathbf{x}^{(k)})^2} \tag{22}$$

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

$$\boxed{\log\left(|\boldsymbol{\Sigma}^{(k)}|\right) = \log\left(|\boldsymbol{\Sigma}^{(k-1)}|\right) + \log\alpha} \tag{23}$$

To give an example, if a measure of good classification is used, we need to compute the decision rule and with the update rules, it can be rewritten as follow

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

It can be noticed that this new formula 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 Figure 3 illustrates the optimization of the selection algorithm.

Nevertheless, 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 $\boldsymbol{\Lambda}$ the diagonal matrix of eigenvalues and $\mathbf{Q}$ the matrix of eigenvectors s.t. $\boldsymbol{\Sigma} = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^t$.

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 ill-estimated 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 the machine maximum precision for a float, we set these eigenvalues to EPS_FLT.

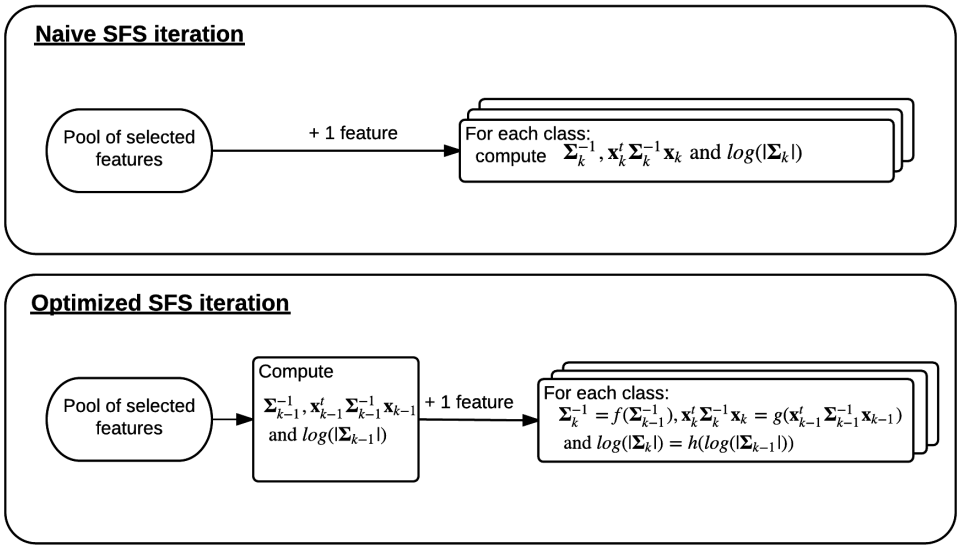


Figure 3: Iteration of sequential forward features selection.

5. Organization

5.1. Objectives

Several outcomes are expected of this internship. First, based on the previous work [3] of M.Fauvel, we want to **reproduce the SFS selection method and develop an upgrade version** corresponding to the floating forward selection described in the previous section. This first part will results in a Python code freely available on Github [2].

Then, the aim is 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** ([6]). 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 [3].

Finally, we want to **test the efficiency of this selection method and compare it to a kernel-based method** ([7]). 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, HoG, morphological profiles, radiometric indexes, ...).

5.2. Planning

As can be expected, the cycle 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

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

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 4 summarized the expected organization of the project.

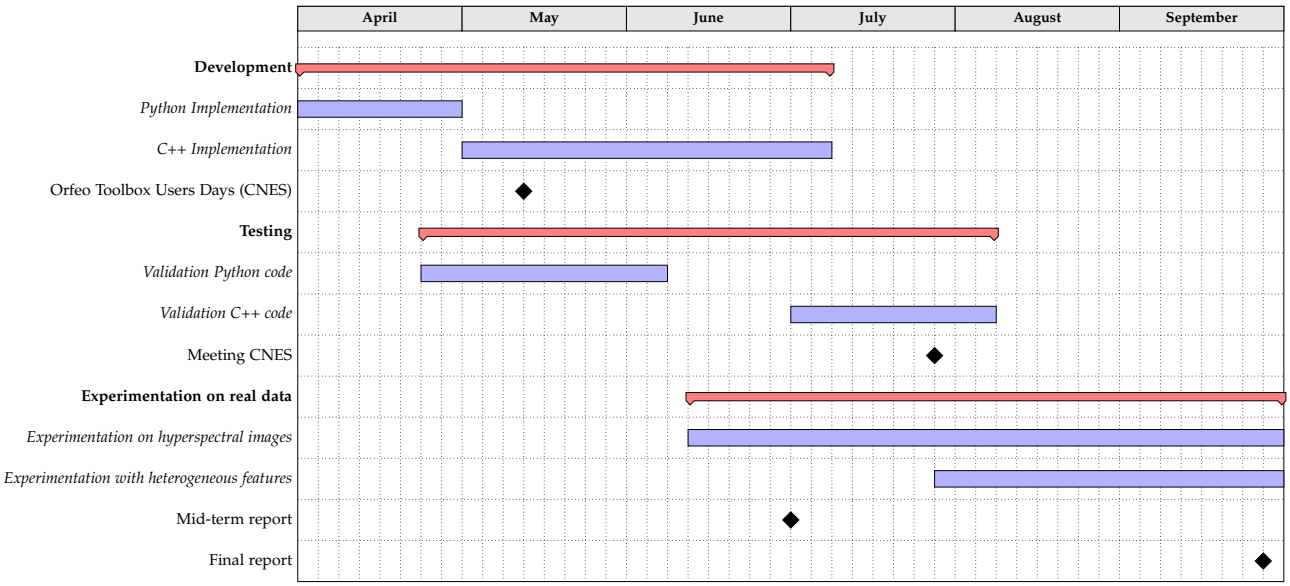


Figure 4: Gantt diagram of the internship.

6. Experimental results about features selection with GMM

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 single dataset presented in the next subsection.

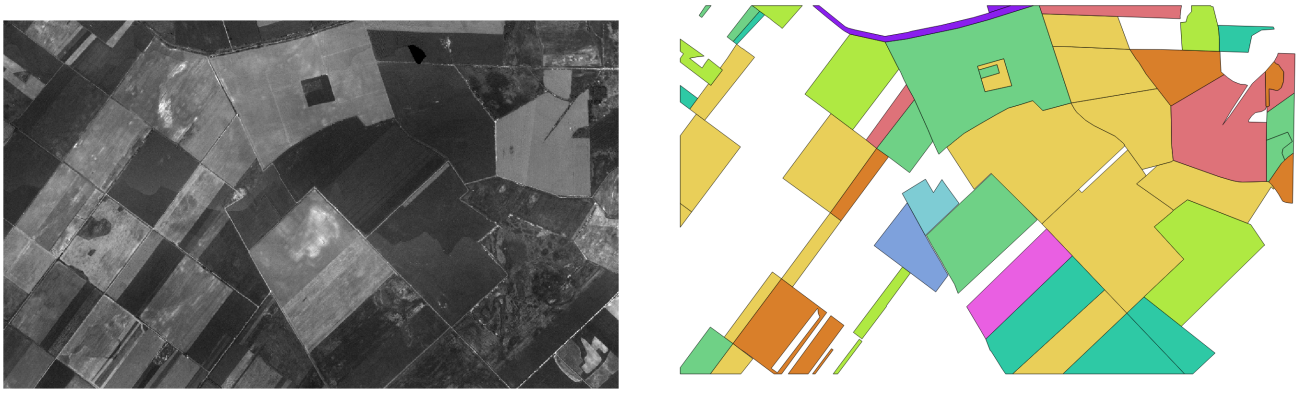
6.1. Aisa dataset

The 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 5 shows one channel of the multispectral image and the shapefile representing groundtruth and Table 6 presents the repartition of the various classes.

6.2. 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.

Figure 7 corresponds to the evolution of the classification rate in function of the number of selected variables with the three metrics introduced before (overall accuracy, Cohen's kappa and mean of F1-score). 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 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.



(a) **(b)**

Figure 5: Aisa dataset: **(a)** a single band of the image, **(b)** groundtruth.

Class	Number of samples
1	136,524
2	61,517
3	30,197
4	17,626
5	18,278
6	7,199
7	10,746
8	23,283
9	2,799
10	4,222
11	4,773
12	26,566
13	9,272
14	3,426
15	2,107
16	3,436

Figure 6: table
Repartition of classes in Aisa dataset.

It is also interesting to see 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 better with more samples and also that the maximum is reached with more selected variables which 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 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 to the influence of criterion function, we can see from Figure 10 the results of classification applied to the validation set for each of criterion presented in Section 2.2. What we expected was that the criterion based on good classification mesure would be better because it

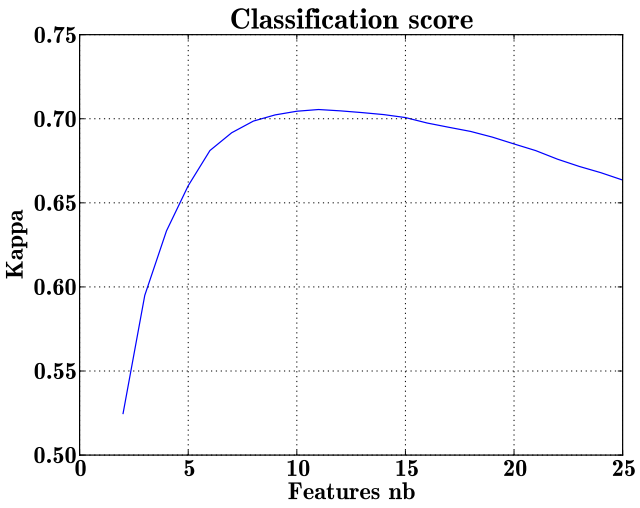


Figure 7: Classification results averaged on 20 trials using forward selection and Jeffries-Matusita distance as criterion with 100 samples by class in training set.

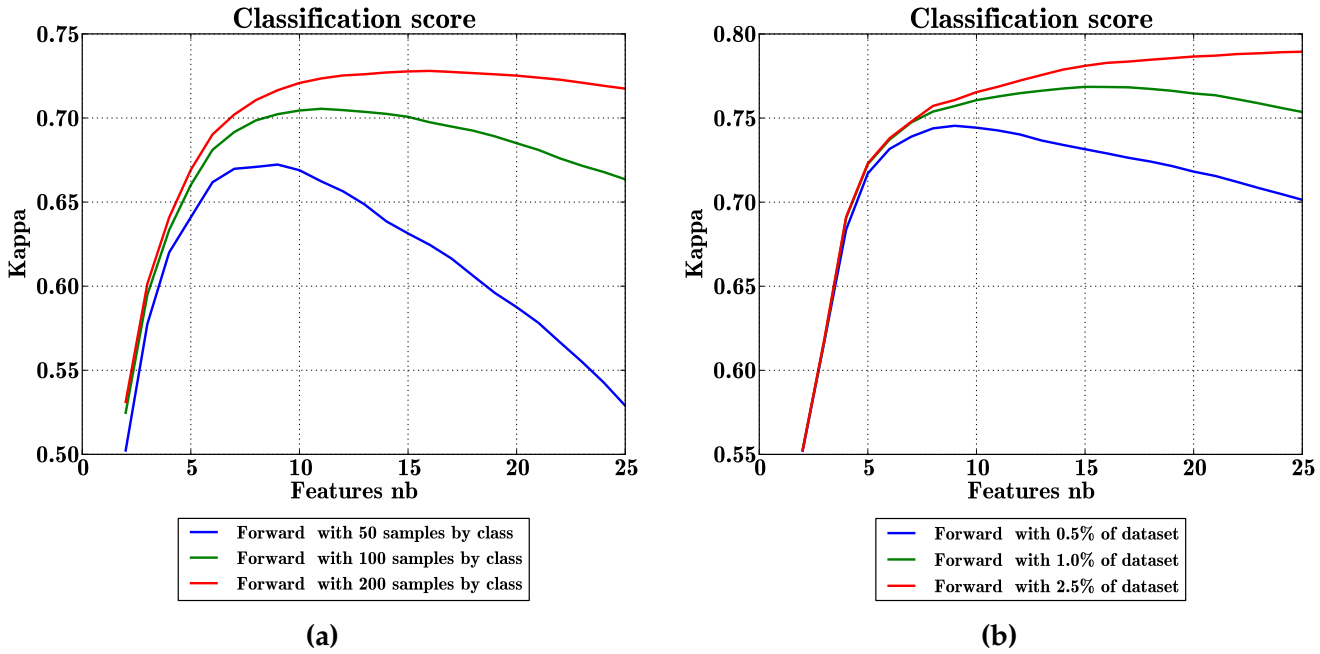


Figure 8: Classification results averaged on 20 trials using forward selection and Jeffries-Matusita distance as criterion: **(a)** with 50, 100, 200 samples by class in training set and **(b)** with 0.5%, 1%, 2.5% of dataset in training set keeping proportion.

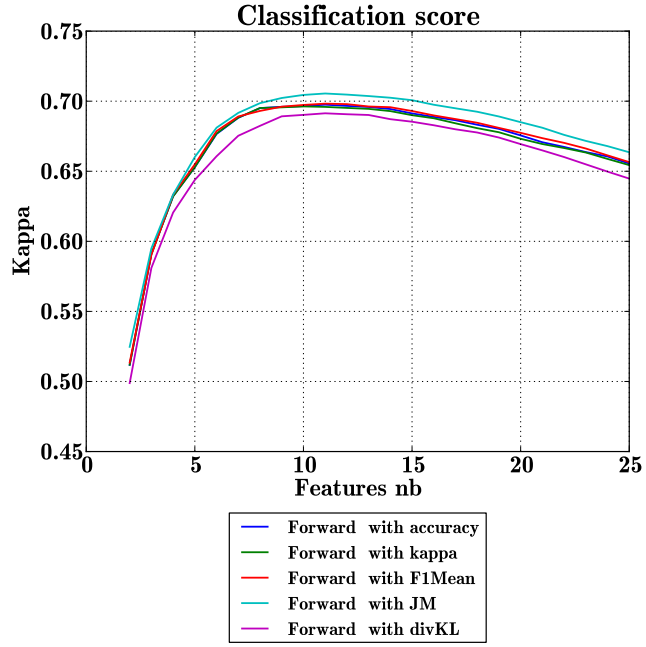
would optimize directly the classification rate and maybe more flexible regarding to the Gaussian assumption. But it appears that for this given dataset, as good results are obtained with Jeffries-Matusita distance.

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 metrics based on distance between distribution (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

(a)

(b)

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



(a)

(b)

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 (=xxxx samples).

Figure 11.

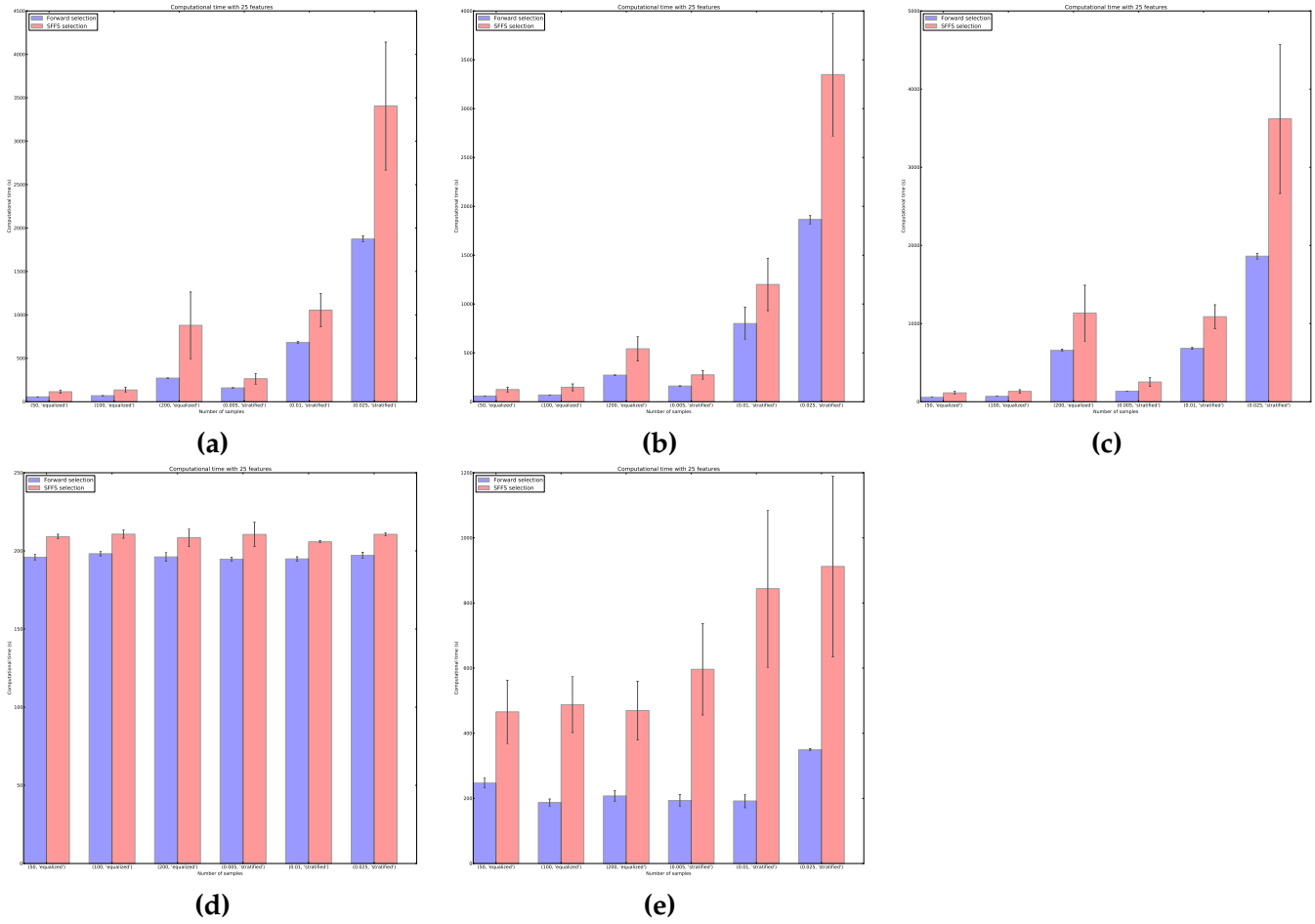


Figure 11: (Figures temporaires en attente de resultats) Processing time for selection with 20 trials : **(a)** using Cohen's kappa, **(b)** using overall accuracy, **(c)** using mean of F1-scores, **(d)** using KL divergence, **(e)** using JM distance.

7. Implementation of the Orfeo Toolbox module

After the calibration of the algorithm with a Python implementation, we aim to realise 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.

7.1. Integration in Orfeo Toolbox

The module developed is available on Github [4] and is actually a fork of the template for remote module furnished by OTB developers.

7.2. 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.

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

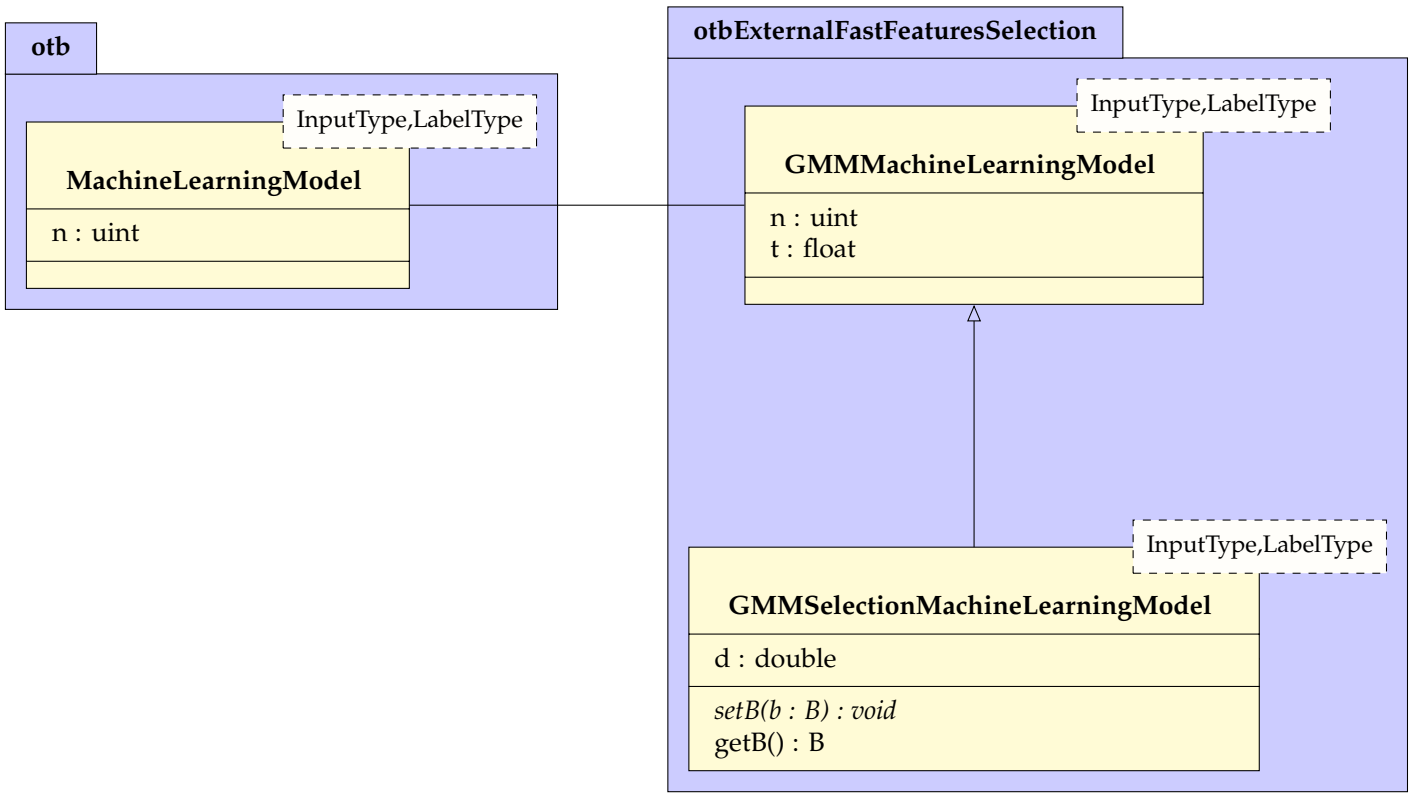


Figure 12: UML digram of the developped OTB remote module.

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 errors when parameters are ill-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;
- 4 precomputed terms used for prediction: eigenvalues of covariance matrices, eigenvectors of

covariance matrices, 2 other terms.

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 12.

7.3. 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.

7.4. Code profiling

A voir, si je trouve un truc lisible a presenter mais ca serait bien...

8. Experimental results comparison to classical classifiers

9. Conclusion

A. Update for cross validation (calculation)

A.1. Mean update

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

$$\boxed{\boldsymbol{\mu}_c^{n_c} = \boldsymbol{\mu}_c^{n_c-\nu_c} + \tfrac{\nu_c}{n_c}(\boldsymbol{\mu}_c^{\nu_c} - \boldsymbol{\mu}_c^{n_c-\nu_c})} \tag{24}$$

$$\boxed{\boldsymbol{\mu}_c^{n_c-\nu_c} = \tfrac{n_c\boldsymbol{\mu}_c^{n_c} - \nu_c\boldsymbol{\mu}_c^{\nu_c}}{n_c-\nu_c}} \tag{25}$$

A.2. Covariance matrix update

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

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

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

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

$$
\boxed{\boldsymbol{\Sigma}_c^{n_c} = \tfrac{n_c-\nu_c-1}{n_c-1}\boldsymbol{\Sigma}_c^{n_c-\nu_c} + \tfrac{\nu_c-1}{n_c-1}\boldsymbol{\Sigma}_c^{\nu_c} + \tfrac{n_c\nu_c}{(n_c-1)(n_c-\nu_c)}(\boldsymbol{\mu}_c^{\nu_c}-\boldsymbol{\mu}_c^{n_c})(\boldsymbol{\mu}_c^{\nu_c}-\boldsymbol{\mu}_c^{n_c})^t} \tag{26}
$$

$$
\boxed{\boldsymbol{\Sigma}_c^{n_c-\nu_c} = \tfrac{1}{n_c-\nu_c-1}((n_c-1)\boldsymbol{\Sigma}_c^{n_c} - (\nu_c-1)\boldsymbol{\Sigma}_c^{\nu_c} - \tfrac{n_c\nu_c}{(n_c-\nu_c)}(\boldsymbol{\mu}_c^{\nu_c}-\boldsymbol{\mu}_c^{n_c})(\boldsymbol{\mu}_c^{\nu_c}-\boldsymbol{\mu}_c^{n_c})^t\ )} \tag{27}
$$

References

[1] 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.

[2] 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.

[3] 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.

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

[5] 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.

[6] 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.

[7] G. Camps-Valls, J. Mooij, and B. Scholkopf, "Remote sensing feature selection by kernel dependence measures," *IEEE Geoscience and Remote Sensing Letters*, vol. 7, no. 3, pp. 587–591, 2010.