Chapitre 7

Manipulations du Master 1 - IM et SIA

7.1 Présentation des manipulations

7.1.1 Introduction

Les travaux pratiques de classification sont répartis sur deux séances de trois heures. Leur but est d'illustrer toutes les étapes rencontrées lors de la résolution d'un problème de classification. La mise en œuvre est prévue sous Python 3. Le séquencement sera le suivant :

TP no1 :
- Etude temporelle et fréquentielle des signaux.
- L'extraction de paramètres pertinents pour la classification envisagée.
- Prétraitement des paramètres en vue de la classification.

TP no2 :
- La mise en œuvre de méthodes de classification :
  * Maximum a posteriori
  * Algorithme des centres mobiles - KMeans
  * Perceptron multicouches - MLP
  * Séparateurs à vastes marges - SVM
  * Forêt d'arbres décisionnels - Random Forest
- L'évaluation de la qualité de la classification obtenue.

Avant d'arriver en séance, vous devez impérativement :
- Avoir assimilé les notions présentées en cours et en TD.
- Avoir préparé les programmes demandés pour consacrer l'essentiel de la séance à l'obtention et l'interprétation des résultats.

Les fichiers nécessaires sont disponibles sur votre espace Moodle

7.2 Séance n^o1 : Analyse des signaux et prétraitement des paramètres

7.2.1 Présentation de la manipulation

Pour la manipulation, vous aurez à votre disposition un ordinateur ave le langage de programmation Python qui vous permettra d'effectuer les traitements envisagés. Les signaux sont générés par le programme `genere.py` et stockés dans le fichier `signaux.txt`. Le programme de lecture (a compléter) est fourni dans le fichier `TPClassif2024.py`.

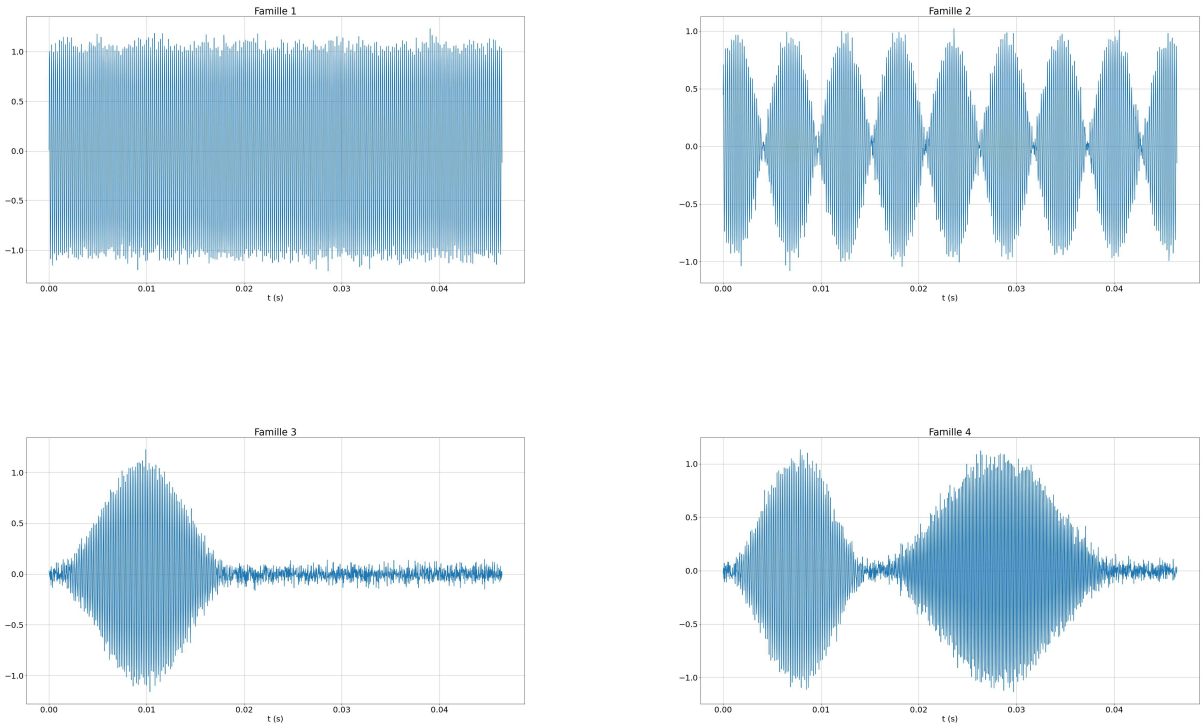

FIGURE 7.1 – Différents types de signaux à reconnaître

Les signaux à reconnaître se répartissent en quatre familles qui sont présentées à la figure n° 7.1. La fréquence d'échantillonnage utilisée est 44100 Hz.

7.2.2 Travail à effectuer

Les titres suivis d'une astérisque doivent être préparés pour la séance de TP. Compte tenu des procédures disponibles et détaillées plus loin, la programmation reste aisée car les programmes sont courts.

Analyse du signal* *partiellement*

L'objet de cette partie est de mettre en évidence des singularités permettant de discerner les familles de signaux entre elles. Pour cela, une étude temporelle puis fréquentielle des signaux sera effectuée. L'examen temporel des signaux montre que l'enveloppe est porteuse d'information. Nous savons que le module de la transformée de Hibert d'un signal à bande étroite correspond à l'enveloppe de celui-ci. Aussi, pour le TP, vous pourrez utiliser les commandes suivantes :

```
import numpy as np
from scipy.signal import hilbert
Enveloppe = np.abs(hilbert(MatriceDonnees))
```

Extraction de paramètres pertinents

En fonction des résultats obtenus précédemment, écrire un programme Python qui permette d'extraire, au moins, trois paramètres représentatifs des signaux de façon automatique.

Prétraitement des données*

Afin d'améliorer les performances lors de la classification, vous réaliserez une Analyse Factorielle Discriminante AFD sur les données tout en réduisant l'espace de représentation à deux paramètres. Afin de pouvoir comparer les deux approches, vous réaliserez aussi une Analyse en Composantes Principales dans les mêmes conditions. Vous prendrez soin de justifier l'intérêt d'utiliser ou non des variables centrées-réduites.

Le sujet du TP continue en page 126.

7.2.3 Deuxième partie : Mise en œuvre et évaluation de méthodes de classification.

Introduction :

Le but de cette partie est de mettre en évidence les performances des différentes méthodes de classification envisagées. Pour les cinq classifieurs, vous évaluerez les performances de chacune des méthodes sur les données issues du prétraitement par Analyse Factorielle Discriminante et par Analyse en Composantes Principales.
Afin d'éviter un comportement similaire à l'apprentissage par cœur, vous partagerez la base d'exemples disponibles en deux parties :

— Une base d'apprentissage comprenant 90% des individus.
— Une base de test comprenant le reste des individus.

Si les individus de la base de test n'ont pas servi à l'apprentissage, il sera possible de tester les capacités de généralisation du système de classification, dans des conditions proches de la phase d'exploitation, en évaluant ses performances sur la base de test.

Mise en œuvre de méthodes de classification.

Travail à effectuer : Le but de cette partie est de mettre en œuvre différentes méthodes de classification disponibles dans le module **sklearn** :

— Maximum a posteriori :

  Une procédure sklearn n'est pas nécessaire. Nous admettrons l'hypothèse d'une répartition gaussienne des paramètres et vous calculerez les probabilités a posteriori $P(X/C_q)$ pour chaque classe pour affecter l'individu X à la classe la plus probable.

— Algorithme des centres mobiles :

```
Classif=sklearn.cluster.KMeans( ... Options ... )
Classif.fit(X_Apprentissage, NoClasses)
y_predict=Classif.predict(X_Test)
```

— Perceptron multicouche :

```
Classif = sklearn.neural_network.MLPClassifier( ... Options ... )
Classif.fit(X_Apprentissage, NoClasses)
y_predict=Classif.predict(X_Test)
```

— Séparateurs à Vastes Marges :

```
Classif = sklearn.svm.SVC( ... options ... )
Classif.fit(X_Apprentissage, NoClasses)
y_predict=Classif.predict(X_Test)
```

— Forêt d'arbres décisionnels :

```
Classif = sklearn.ensemble.RandomForestClassifier( ... options ... )
```

```
Classif.fit(X_Apprentissage, NoClasses)
y_predict=Classif.predict(X_Test)
```

Evaluation des performances et recherche des performances optimales.

Évaluation des performances : Afin de pouvoir comparer objectivement les différentes solutions mises en œuvre, vous devrez définir (ou choisir) une méthode d'évaluation du taux de réussite sur la reconnaissance des individus de la base de test.
Il est conseillé de choisir les individus de la base de test au hasard, les résultats sont donc susceptibles de varier d'une exécution à une autre. Aussi, vous devrez fournir la moyenne des valeurs obtenues sur plusieurs exécutions. L'évaluation des performances peut être réalisée grâce à deux procédures du module `sklearn` :

— Matrice de confusion :

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay

y_pred = clf.predict(X_test)
cm = confusion_matrix(y_reference, y_pred)
cm_display = ConfusionMatrixDisplay(cm).plot()
```

— Taux de réussite :

```
from sklearn.metrics import accuracy_score

y_pred = clf.predict(X_test)
accuracy_score(y_reference, y_pred)
```

Recherche des performances optimales : A partir du taux de réussite calculé à l'étape précédente, répondez aux questions qui se posent lors de la mise en œuvre d'un système de reconnaissance automatique :

- Comparaison des performances obtenues pour un prétraitement par AFD, par ACP ou sans prétraitement.
- Choix du nombre de paramètres retenus pour la description des individus.
- Choix du nombre d'axes conservés pour l'AFD ou l'ACP.
- Choix du nombre de classes en fonction de la méthode de classification.
- Interprétation des résultats et conclusions : Dans cette partie, vous proposerez, en justifiant votre choix, votre solution, c'est à dire le choix et le prétaitement des paramètres, la méthode de classification retenue ainsi que les performance obtenues.

7.2.4 Compte-rendu

Enumérez les points délicats que vous avez rencontrés tout au long de la manipulation et répondez aux questions posées dans le sujet.
L'interprétation des résultats est une étape essentielle dans la résolution de tout problème, aussi, vous veillerez à expliquer en détail tous les résultats obtenus.

7.3 Présentation des procédures Python mises à disposition.

Les procédures utiles pour la mise en œuvre de l'AFD et l'ACP sont contenues dans le module décrit par `ModuleTPClassif.py`. Voici, ci-après, le détail de chacune d'elles.

Normalisation des variables

Cette fonction permet d'obtenir des variables centrées et réduites.

```
def CalculerIndividusCentresReduits(Individus, CentresGravite) :
#-------------------------------------------------------------------------------
#
# Calcul des individus centrés réduits
#
#-------------------------------------------------------------------------------
# Entree : Individus                [NbrIndividus x NbrParametres]
#          CentresGravite           [NbrClasses   x NbrParametres]
#
# Sortie : IndividusCentresReduits  [NbrIndividus x NbrParametres]
#-------------------------------------------------------------------------------
```

Calcul des centres de gravité

Cette fonction permet de calculer les centres de gravité des classes telles quelles sont définies par la variable `classes`.
Attention : La classe zéro est réservée pour désigner l'ensemble des individus. Aussi, vous devez numéroter les classes en commençant par le numéro 1.

```
def CalculerCentresGravite(Individus, NoClasses) :
#-------------------------------------------------------------------------------
#
# Calcul des centres de gravité de chaque classe.
#
#-------------------------------------------------------------------------------
# Entree : Individus               [NbrIndividus x NbrParametres]
#          NoClasses               [NbrIndividus]
#
# Sortie : CentresGravite          [NbrClasses   x NbrParametres]
#          Nb : CentreGravite[0] =  Centre gravite global
#-------------------------------------------------------------------------------
```

Calcul des variances

Cette fonction permet de calculer les valeurs des variances telles quelles ont été définies en cours.

```
def CalculerVariances(Individus, NoClasses, CentresGravite) :
#-------------------------------------------------------------------------------
#
# Calcul des variances Totale, Intraclasses et Interclasses
#
#-------------------------------------------------------------------------------
# Entree : Individus          [NbrIndividus x NbrParametres]
#          NoClasses          [NbrIndividus x 1 ]
#          CentresGravite     [NbrClasses   x NbrParametres]
#
# Sortie : VT = Variance totale       [1 x 1]
#          VA = Variance intraclasses [1 x 1]
#          VE = Variance interclasses [1 x 1]
#-------------------------------------------------------------------------------
```

Calcul des matrices de covariance

Cette fonction permet de calculer les matrices de covariances telles qu'elles ont été définies en cours.

```
def CalculerMatricesCovariance(Individus, NoClasses, CentresGravite):
#-------------------------------------------------------------------------------
#
# Calcul des matrices de covariance Totale, Intraclasses et Interclasses
#
#-------------------------------------------------------------------------------
# Entree : Individus          [NbrIndividus x NbrParametres]
#          NoClasses          [NbrIndividus x 1 ]
#          CentresGravite     [NbrClasses   x NbrParametres]
#
# Sortie : CT = Matrice de covariance totale       [NbrParametres x NbrParametres]
#          CA = Matrice de covariance intraclasses [NbrParametres x NbrParametres]
#          CE = Matrice de covariance interclasses [NbrParametres x NbrParametres]
#-------------------------------------------------------------------------------
```

Calcul des matrices de covariance de chacune des classes

Pour la méthode du maximum a posteriori, il est necessaire de calculer les densités de probabilité qui dépendent de la matrice de covariance de la classe considérée.

$$p(\underline{X}/C_q) = \frac{1}{(2\,\pi)^{\frac{p}{2}}\ \sqrt{\left|MC_q\right|}}\ e^{-\frac{1}{2}\left(\underline{x}-\underline{\overline{x}}_q\right)^T MC_q^{-1}\left(\underline{x}-\underline{\overline{x}}_q\right)}$$

Cette fonction retourne une liste contenant les matrices de covariance inverse de chacune des classes.

```
def CalculerMatricesCovarianceInverseClasses(Individus, NoClasses, CentresGravite):
#-------------------------------------------------------------------------------
#
# Calcul des matrices de covariance inverse de chacune des classes
#
#-------------------------------------------------------------------------------
# Entree : Individus          [NbrIndividus x NbrParametres]
#          NoClasses          [NbrIndividus x 1 ]
#          CentresGravite     [NbrClasses   x NbrParametres]
#
# Sortie : Cq_inv = Liste des matrices de covariance inverse des classes
#               Liste de dimensions  Q x [NbrParametres x NbrParametres]
#-------------------------------------------------------------------------------
```

Visualisation des nuages de points

Cette fonction permet d'afficher les individus en fonction de leurs paramètres. Si rien n'est précisé, les graphes correpondent aux individus projetés sur le plan défini par les paramètres pris deux à deux. Toutes les combinaisons sont passées en revue dans la limite de 10 graphes. Pour augmenter le nombre de graphes, il suffit d'appeler la procédure avec le paramètre MaxGraphes :

```
PresenterClasses(Individus, NoClasses, Titre, MaxGraphes=50):
```

Lorsqu'il y a trop de graphes pour les afficher tous, il est possible d'en choisir un en précisant le numéro du paramètre sur l'axe des abscisses : *ParamX* et sur l'axe des ordonnées : *ParamY*, comme, par exemple :

```
PresenterClasses(Individus, NoClasses, Titre, ParamX=2, ParamY=5):
```

```
def PresenterClasses(Individus, NoClasses, Titre, CentresGravite=[], NomParametres=[],
                     MaxGraphes=10, ParamX=0, ParamY=0):
#-------------------------------------------------------------------------
#
#     Présentation des individus et le centre de gravité des classes
#             à partir des paramètres pris deux par deux
#
#-------------------------------------------------------------------------
# Entree : Individus            [NbrIndividus x NbrParametres]
#          NoClasses            [NbrIndividus x 1 ]
#          Titre                [Chaine de caractères ]
#          CentresGravite       [NbrClasses x NbrParametres] - Optionnel
#          MaxGraphes           Scalaire                     - Optionnel
#          ParamX               Scalaire                     - Optionnel
#          ParamY               Scalaire                     - Optionnel
#
# Sortie : Visualisation graphique
#
#-------------------------------------------------------------------------
```