Chapitre 6

Manipulations du Master 1 - SRI

6.1 Présentation des manipulations

Les travaux pratiques de classification comportent deux manipulations réparties sur quatre séances de deux heures. Le premier TP porte sur la Reconnaissance de symboles phonétiques, son but est d'illustrer toutes les étapes rencontrées lors de la résolution d'un problème de classification.

Séance n^o1 :
- Étude temporelle et fréquentielle des signaux.
- Calcul des coefficients cepstraux.
- Programmation d'une Analyse Factorielle Discriminente (AFD).

Séances n^o2 et 3 :
- Mise en œuvre de méthodes de classification.
  - Maximum a posteriori
  - Algorithme des centres mobiles.
  - Perceptron multicouche.
  - Séparateurs à Vastes Marges.
  - Forêt d'arbres décisionnels.
- Évaluation de la qualité de la classification obtenue pour les cinq méthodes de classification utilisées.
- Comparaison des performances obtenues pour un prétraitement par AFD, par ACP ou sans prétraitement.
- Choix du nombre d'axes conservés pour l'AFD.
- Choix du nombre de coefficients cepstraux retenus pour la description des individus.
- Choix du nombre de classes en fonction de la méthode de classification.
- Interprétation des résultats et conclusions.

Le second TP porte sur la mise en œuvre d'un réseau Bayésien.

Séance n^o4 :
- Modélisation du problème et définition du réseau Bayésien.
- Mise en œuvre du réseau.

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.

6.2 TP n^o1 : Reconnaissance de symboles phonétiques

6.2.1 Première partie : Analyse des signaux et prétraitement des paramètres

Présentation des signaux à reconnaître

Vous trouverez, dans le répertoire `Signaux`, 1000 fichiers au format `wav` contenant 1024 échantillons de sons correspondant à des voyelles au sens de la phonétique :

Symbole phonétique internationnal	Exemples de mots	Nom du fichier	indices
[a]	**a**rbre, f**e**mme, viol**e**mment	aa ? ?.wav	00 à 99
[e]	pi**ed**, th**é**, qu**ai**, m**es**	ee ? ?.wav	00 à 99
[ɛ]	f**ê**te, l**ai**t, m**e**rci, No**ë**l	eh ? ?.wav	00 à 99
[ø]	j**eu**, bl**eu**, je**û**ner, cr**eu**se	eu ? ?.wav	00 à 99
[i]	f**i**l, p**y**jama, **i**dée, m**ee**ting	ii ? ?.wav	00 à 99
[œ]	p**eu**r, fl**eu**r, cu**ei**llir, **oe**il	oe ? ?.wav	00 à 99
[ɔ]	p**o**rte, **o**tarie, y**a**cht, alc**oo**l	oh ? ?.wav	00 à 99
[o]	sh**ow**, b**eau**, **au**diteur, diplô**me**	oo ? ?.wav	00 à 99
[u]	c**ou**p, f**oo**tball, tr**ou**, a**oû**t	uu ? ?.wav	00 à 99
[y]	p**u**nir, él**u**, e**û**mes, aig**u**ë	yy ? ?.wav	00 à 99

Travail à effectuer

Lecture des fichiers : Pour réaliser la lecture des fichiers, vous pourrez utiliser les modules `os` et `scipy` pour :

Récupérer les noms des fichiers dans une boucle :
`for NomFichier in os.listdir(CheminFichiers)`

Récupérer la fréquence d'échantillonnage et les échantillons contenus dans un fichier au format `wav` : `(Fe, Echantillons)=scipy.io.wavfile.read(CheminFichiers+NomFichier)`

Récupérer le numéro de classe :

```
Prefixe=['aa', 'ee', 'eh', 'eu', 'ii', 'oe', 'oh', 'oo', 'uu', 'yy']
NumeroClasse=Prefixe.index( NomFichier[0:2] )
```

Analyse du signal : 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. Le calcul de la Denstité Spectrale de Puissance s'effectue grâce aux commandes :

```
matplotlib.figure(1)
matplotlib.plot(Echantillons)
matplotlib.title('Représentation temporelle du signal')

tf_signal=numpy.fft.fft(Echantillons)
dsp=numpy.multiply(tf_signal, np.conj(tf_signal))
matplotlib.figure(2)
matplotlib.plot(dsp)
matplotlib.title('Densité Spectrale de Puissance')
matplotlib.show()
```

Calcul des coefficients cepstraux - Mel Frequency Cepstral Coefficients : Lorsqu'un signal est issu d'une opération de filtrage $y(t) = x(t) * h(t)$, il peut être intéressant de séparer le signal d'entrée de la réponse impulsionnelle du filtre. Si l'on calcule la transformée de Fourier (passage d'un produit de convolution à un produit), puis le logarithme (passage d'un produit à une somme) et enfin la transformée de Fourier inverse (retour dans le domaine temporel mais sous forme d'une somme), il vient :

$$Y(f) = X(f)\,H(f) \implies \log Y(f) = \log X(f) + \log H(f) \implies c(\tau) = \mathcal{F}^{-1}\{\log X(f)\} + \mathcal{F}^{-1}\{\log H(f)\}$$

Pour peu que les supports fréquentiels de $X(f)$ et $H(f)$ ne soient pas les mêmes, les deux informations seront séparées. Bien que le calcul soit un peu plus complexe car il est tenu compte de la sensibilité de l'oreille humaine selon une échelle logarithmique (échelle de Mel), c'est ce principe qui est mis en œuvre pour le calcul des coefficients cepstraux. Dans le domaine du traitement de la parole, la pertinence des coefficients mfcc est universellement reconnue, nous utiliserons donc, principalement, ces coefficients, leur valeur est calculée par :

```
VecteurCoefficients = features.mfcc(Echantillons, Fe, ... Options ... )
```

La procédure est écrite pour fournir un flot continu de `numcep` coefficients calculés sur une fenêtre de largeur `winlen` avec un décalage de `winstep`. Dans notre cas et pour obtenir un seul jeu de coefficients par fichier, vous devrez fixer `winlen` et `winstep` égaux à la durée du fichier `wav`, c'est à dire *NombreEchantillons*/*Fe*.

Il est conseillé de ranger les résultats dans une matrice $N \times P$ où :
N : Nombre d'individus (Nombre de fichiers utilisés pour l'apprentissage)
P : Nombre de paramètres (Nombre de coefficients mfcc)
De même le numéro de la classe pourra être rangé dans un vecteur de N lignes.

Prétraitement des données : Afin d'améliorer les performances lors de la classification, il est utile de réaliser une Analyse Factorielle Discriminente AFD sur les données. Vous devrez écrire le programme sous Python v3 pour réaliser l'AFD t'elle qu'elle a été présentée en travaux Dirigés. Vous prendrez soin de justifier l'intérêt ainsi que le cadre de l'utilisation de variables centrées-réduites pour la suite de la manipulation, par exemple pour l'utilisation de la distance Euclidienne.

Le sujet du TP continue en page 126.

6.3 TP n°2 - Réseaux Bayésiens

6.3.1 Introduction

La manipulation vise à estimer le risque qu'il y aurait à assurer un véhicule. Le problème est complexe et sera simplifié pour pouvoir être traité dans le cadre d'une séance de TP. Les données font partie des exemples classiques et sont disponibles sur internet à l'adresse :

`https://www.bnlearn.com/bnrepository/discrete-medium.html#insurance`

Les paramètres sont au nombre de 27, tous ne sont pas pertinents pour notre application car il n'y a pas encore eu d'accident ou de vol :

GoodStudent (Elève sérieux) : False ou True.
Age (Age) : Adolescent, Adult et Senior.
SocioEcon (Status socio-économique) : Prole, Middle, UpperMiddle et Wealthy.
RiskAversion (Aversion au risque) : Psychopath, Adventurous, Normal et Cautious.
VehicleYear (Age du véhicule) : Current et older.
ThisCarDam (Dégâts au véhicule / en cas d'accident) : None, Mild, Moderate et Severe.
RuggedAuto (robustesse du véhicule) : EggShell, Football et Tank.
Accident (Gravité de l'accident / en cas d'accident) : None, Mild, Moderate et Severe.
MakeModel (Type de véhicule) : SportsCar, Economy, FamilySedan, Luxury et SuperLuxury.
DrivQuality (Type de conduite) : Poor, Normal et Excellent.
Mileage (Kilométrage) : FiveThou, TwentyThou, FiftyThou et Domino.
Antilock (ABS) : False ou True.
DrivingSkill (compétances du conducteur) : SubStandard, Normal et Expert.
SeniorTrain (Formation personnes agées) : False ou True.
ThisCarCost (Coût d'achat du véhicule) : Thousand, TenThou, HundredThou et Million.
Theft (vol / en cas de vol) : False ou True.
CarValue (Valeur actuelle du véhicule) : FiveThou, TenThou, TwentyThou, FiftyThou et Million.
HomeBase (Type de voisinage) : Secure, City, Suburb et Rural.
AntiTheft (Système anti-vol) : False ou True.
PropCost (Rapport du coût entre les véhicules / en cas d'accident) : Thousand, TenThou, HundredThou et Million.
OtherCarCost (Coût d'achat de l'autre véhicule / en cas d'accident) : Thousand, TenThou, HundredThou et Million.
OtherCar (Véhicule tiers / en cas d'accident) : False ou True.
MedCost (Coût des soins médicaux / en cas d'accident) : Thousand, TenThou, HundredThou et Million.
Cushioning (Système d'amortisseurs) : Poor, Fair, Good et Excellent.
Airbag (airbag) : False ou True.
ILiCost (Coût de l'expertise / en cas d'accident) : Thousand, TenThou, HundredThou et Million.
DrivHist (Nombre d'accidents) : Zero, One et Many.

6.3.2 Partie n^o1 : Modélisation du problème

Afin de simplifier le problème, vous pourrez adopter le réseau Bayésien, ci-après, que vous compléterez. Il s'agit de résumer, pour le couple conducteur/véhicule, tous les paramètres pertinents en quatre informations :

— Comportement du conducteur (CC) : Doit refléter le risque que représente le conducteur.
— Etat de santé du conducteur (SC) : Doit refléter les coûts éventuels de frais médicaux en cas d'accident.

— Coût du véhicule (CV) : Doit refléter les coûts éventuels de remise en état en cas d'accident.
— Type de véhicule (TV) : Doit refléter les risques qu'il y a à conduire ce véhicule.

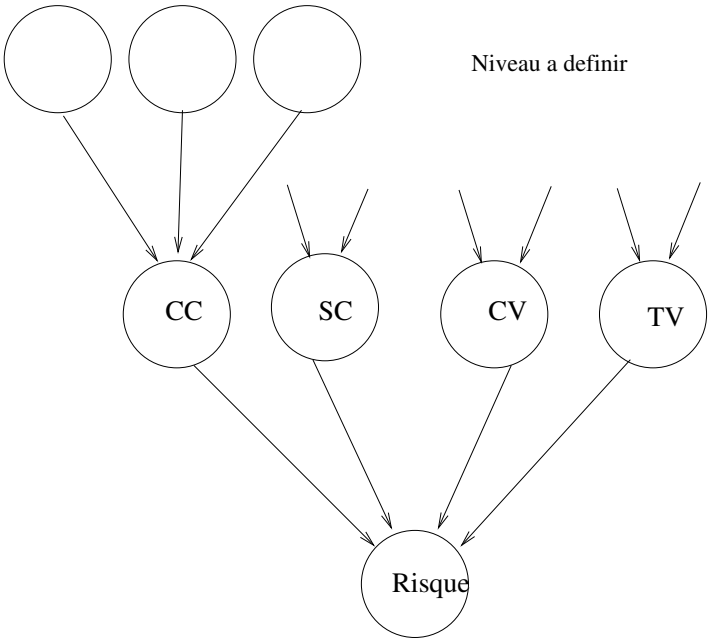

FIGURE 6.1 – Réseau Bayesien à compléter

Travail à effectuer :
— Vous devrez associer les paramètres pertinents du fichier de données à chacune des informations décrites ci-dessus (CC, SC, CV, TV).
— En fonction des association précédentes, vous devrez définir le tableau correspondant à chacune des quatre informations (CC, SC, CV, TV).
— A partir de ces quatre informations (CC, SC, CV, TV), vous définirez le tableau des probabilités de la variable correspondant au risque lié au contrat proposé (Raisonable, Important).

Vous prendrez soin de remplir les tableaux avec des valeurs de probabilité "logiques", une vraie expertise prendrait beaucoup trop de temps.

6.3.3 Partie n^o2 : Mise en œuvre et inférences

Implémentez le réseau préférentiellement sous python, mais vous êtes libre d'utiliser tout autre logiciel pour ce faire. Une fois fait, vous serez à même de répondre aux questions que peut se poser l'assureur :

— Pour un couple Conducteur/Véhicule à qui un contrat d'assurance a été proposé, quelle est la probabilité pour que le conducteur soit « passionné de vitesse » ?
— Pour un conducteur : Sérieux de plus de 50 ans sans antécédent d'accident, conduisant un véhicule de type « luxe » dont le coût est très élevé, qui sollicite un contrat d'assurance, quel est le risque pour l'assureur ?

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
#
#-------------------------------------------------------------------------
```