Pràctica d'Àudio

Processament digital d'àudio i parla

Curs 2020-2021

Enginyeria La Salle

Índex

1. Introducció

L'objectiu d'aquesta activitat pràctica és realitzar una primera experiència de **disseny d'un sistema de classificació automàtica d'àudio**, usant llibreries i datasets ja existents.

Abans de començar la pràctica cal que tingueu instal·lada la versió de Matlab més recent, que inclogui les llibreries següents: Signal Processing, Statistics and Machine Learning, Audio Toolbox. Recordeu que el Matlab té una ajuda molt potent (*Help*), amb el que podeu consultar qualsevol funció així com les diferents formes de crida, i els valors que poden prendre tant els paràmetres d'entrada com les seves sortides. A més, es proporcionen molts exemples d'ús que es poden reproduir per línia de comandes abans d'incloure cap funció per a experimentar.

En primer lloc, treballarem amb col·leccions de dades d'àudio disponibles per la comunitat científica, amb les quals en farem una caracterització usant **atributs freqüencials i cepstrals** per poder representar trames (a curt termini o a mig/llarg termini) com a vectors que ens permetran obtenir posteriorment relacions amb categories associades al tipus de so. Concretament, treballarem la problemàtica de categoritzar escenes acústiques, és a dir, poder obtenir de forma automàtica la categoria del tipus d'entorn a on hem gravat l'àudio a partir de l'anàlisi d'aquest usant els atributs seleccionats. A part d'haver de seleccionar uns o altres atributs per realitzar l'anàlisi del senyal, també haurem de definir aspectes com és la durada i tipus de finestra a usar per al seu càlcul. La seva particularització pot millorar o empitjorar el resultat del posterior aprenentatge artificial, fet pel qual convé estudiar diferents configuracions en el procés d'exploració del tipus d'anàlisi.

En segon lloc, farem ús de **tècniques de classificació** i avaluarem el seu funcionament amb els conjunts de dades proporcionats, i perseguirem realitzar diversos estudis per concloure quines configuracions de paràmetres i classificadors són les més òptimes en les condicions que plantegeu. Per això, es tracta d'una pràctica oberta en el sentit de no donar-vos opcions concretes de prova, tot donant-vos força opcions a configurar tant pel que es refereix a descriptors del senyal d'àudio com a classificadors en els vostres estudis.

En l'apartat 2 se us presenta la base de dades que usareu en els estudis, i com configurar-la per a poder treballar-hi amb el Matlab. En l'apartat 3 se us donen indicacions de les llibreries de funcions que podeu usar en el disseny del vostre codi Matlab, tant d'extracció de característiques com de classificació. En l'apartat 4 teniu l'enunciat de la pràctica, o sigui, les indicacions sobre la funció principal que heu de programar, i els estudis que heu de realitzar amb el codi i la base de dades, i a l'apartat 5 els detalls de com realitzar el lliurament.

2. Base de dades

L'activitat pràctica la realitzarem sobre una base de dades de paisatges acústics, concretament la TUT Acoustic scenes 2017 disponible a l'enllaç aquest[1], formada per 4680 fitxers WAV de 10 segons cadascun. Es tracta d'una base de dades acústica composada per 15 categories de paisatges acústics (4 associats a interiors en vehicles – bus, vehicle cotxe privat, tren i tramvia –, 6 associats a entorns dins d'edificis o *indoor* – cafè/restaurant, botiga de queviures, llar privada, biblioteca, estació de metro, oficina/empresa –, i 5 associats a entorns exteriors o *outdoor* – centre cuitat, camí forestal, platja vora d'un llac, àrea residencial, parc urbà). La descripció detallada del contingut de la base de dades la podeu llegir en el fitxer *README.html* dins de *TUT-acoustic-scenes-2017-development.doc.zip*.

[1] Necessitareu uns 11.5G d'espai en disc per poder descarregar tots els fitxers de la base de dades.

2.1 Organització en carpetes de categories (*organize_dataset.m*)

Per poder preparar la base de dades per treballar adequadament és imprescindible que genereu una funció de tipus *script* de Matlab que anomenareu *organize_dataset.m*, i que s'encarregarà d'automatitzar el procés d'assignar els fitxers d'àudio en subcarpetes (1 subcarpeta per categoria). Com veureu més endavant, és necessari que poseu els fitxers de cada categoria en un subdirectori diferent, i que el nom d'aquest coincideixi amb el de la categoria, que seran els noms de classe especificats dins del fitxer *meta.txt*.

Com veureu al descarregar-vos tots els ZIPs, el fitxer *meta.txt* contingut dins de *TUT-acoustic-scenes-2017-development.meta*, conté les metadades de la base de dades, és a dir, les correspondències entre els fitxers WAV de la base de dades i les 15 classes de paisatges acústics definits per la taxonomia[2].

La lectura del fitxer *meta.txt* la podeu realitzar fent ús de la funció *readtable.m*, concretament:

```
T = readtable(filePath,'Delimiter','\t','ReadVariableNames',false);
```

llegirà la fitxer *filePath* (conté la ubicació i el nom del fitxer *meta.txt*) tenint en compte que els delimitadors de columna són tabuladors i que la primera fila no conté els noms de la taula, i retornarà en la variable `T` una estructura de tipus taula amb els valors de cada fila i columna del fitxer *meta.txt*.

Per accedir al nom de fitxer i categoria associada de la fila *r* (essent *r* un valor enter entre 1 i *size(T,1)*) de la taula *T* ho faríem:

```
filename = cell2mat(T{r,1});
category = cell2mat(T{r,2});
```

Per disposar de les 15 etiquetes de categories podeu fer la crida següent, que us retornarà un cell array amb els 15 noms de categories.

```
categories = unique(T{:,2});
```

Igualment, el fitxer *error.txt* que trobareu dins de *TUT-acoustic-scenes-2017-development.error.zip*, conté la llista de fitxers que per un o altre motiu contenen errors ocorreguts en el procés de gravació, i que haurien de quedar fora de l'anàlisi. Seguint les mateixes recomanacions anteriors, podeu fer la lectura d'aquest fitxer i generar la taula corresponent en Matlab.

Per generar 15 carpetes amb els noms de categories ho podeu fer usant iteradors (*for*) i fent ús de la funció *mkdir.m*. D'altra banda, per assignar cada fitxer a una o altra carpeta, useu també un iterador, de forma que en cada iteració del bucle feu tant el procés d'assignació com la conversió del fitxer original de format estèreo (2 canals) a mono (1 canal), ja que la informació estèreo no la usarem en aquesta activitat:

- Llegiu el fitxer d'àudio original amb la funció *audioread.m*, obtenint tant la matriu *x* de L mostres per 2 (2 canals) com la freqüència de mostratge *fs*.
- Convertiu el senyal d'estèreo a mono sumant les dues columnes i dividint per 2:

```
x = 0.5*sum(x,2);
```

[2] **Nota**: Us recomanem que primer de tot editeu aquest fitxer i canvieu la categoria "cafe/restaurant" per "cafè-restaurant" amb un *reemplaçar*, ja que després posareu els àudios d'aquesta categoria i de totes les altres en subcarpetes (1 per categoria) i el separador "/" no el podreu usar.

- Exporteu el senyal mono generat al fitxer destí ubicat a la carpeta corresponent en funció de la categoria associada al fitxer, usant la funció *audiowrite.m*.

Teniu en compte que només s'hauran de moure els fitxers que no tinguin errors, pel que és recomanable que dins de *l'script* definiu una petita funció amb capçalera

```
e = searchForErrors(filename,errors)
```

que busqui el fitxer amb nom `filename` dins del cell array `errors` que conté la primera columna de la taula associada al fitxer *errors.txt*. Aquesta cerca també la podeu fer amb un iterador i la crida a la funció `strcmp(filename,errors{n})` per comparar el nom del fitxer amb la casella n-èssima del cell array *errors*. D'aquesta forma, si es troba coincidència, podem retornar la variable booleana el amb valor *true* i no realitzar el moviment del fitxer a la carpeta de categoria corresponent, o si fer-lo en cas contrari (si no s'ha trobat coincidència).

3. Funcions de Matlab

A continuació se us donen algunes indicacions de les llibreries de funcions que usarem en aquesta activitat.

3.1 Col·leccions de dades

La funció *audiodatastore.m* permet manipular grans bases de dades sense necessitat que Matlab les carregui de forma explícita en memòria. Per exemple, si fem la crida:

```
ads = audioDatastore(datasetFolder, ...
    'IncludeSubfolders',true, ...
    'FileExtensions','.wav', ...
    'LabelSource','foldernames');
```

Matlab llegirà la carpeta *datasetFolder* i crearà un magatzem de dades d'àudio a partir de l'anàlisi de tots els fitxers d'àudio (en format WAV) continguts en les subcarpetes que contingui aquesta carpeta, i associarà el nom de les subcarpetes a la categoria associada a l'àudio contingut. Una vegada creat el magatzem podrem accedir a les propietats d'aquest, per exemple, *ads.Labels* ens retornarà un vector de N valors categòrics amb el nom de la classe de cadascun dels N fitxers analitzats, mentre que *ads.File*s retornarà el nom de cadascun dels N fitxers de la base de dades.

Usant la crida a la funció *tall.m* sobre l'objecte que retorna la funció *audiodatastore.m* (*ads* en la crida d'exemple) es crearà un cell array amb els N àudios de la base de dades però amb la particularitat que aquest **no s'avaluarà fins que sigui necessari**. Aquesta forma de treballar en Matlab permet fer aplicacions amb grans bases de dades (Big Data, etc.) sense necessitat d'haver de tenir totes les dades carregades en memòria. És a dir, disposem d'una estructura que està preparada per a accedir a les dades, però realment aquestes no es carregaran fins que l'aplicació que estem fent ho requereixi. Per exemple, si fem la crida següent,

```
adsTall = tall(ads);
```

es crearà el cell array *adsTall* que es podrà avaluar o instanciar més endavant.

També disposem de la funció *splitEachLabel.m*, que ens permetrà dividir la base de dades en diverses subcoleccions, **molt útil de cara a realitzar processos de validació creuada amb *k* iteracions (*k-fold cross-validation*)**. Per exemple, si volem dividir un *ads* en 4 subconjunts que continguin cadascú el 25%de les dades (p.ex. 4-fold cross-validation), ho podrem fer invocant:

```
[ads1, ads2, ads3, ads4] = splitEachLabel(ads,0.25,0.25,0.25);
```

De forma *ads1* que contindrà el primer 25% dels fitxers de cada classe o label, mentre que les altres tres variables contenen els següents 25% de dades. D'aquesta forma, ens assegurem que els subconjunts de sortida contenen la mateixa proporció relativa de classes que el conjunt original, evitant així desvirtuar els processos d'entrenament o de test dels diferents mètodes d'aprenentatge artificial que vulguem provar.

És important mencionar que quan comenceu a realitzar proves potser us interessi no usar totes les dades, ja que cada cop que executeu les funcions es requerirà molt temps de CPU. Per poder realitzar petites proves amb menys dades, us recomanem que comenceu fent una subselecció de totes les dades, fent ús de la mateixa funció *splitEachLabel.m* però només per quedar-vos amb una petita porció de totes les dades (p.ex. 5%-10%).

3.2 Extracció de característiques o atributs d'àudio

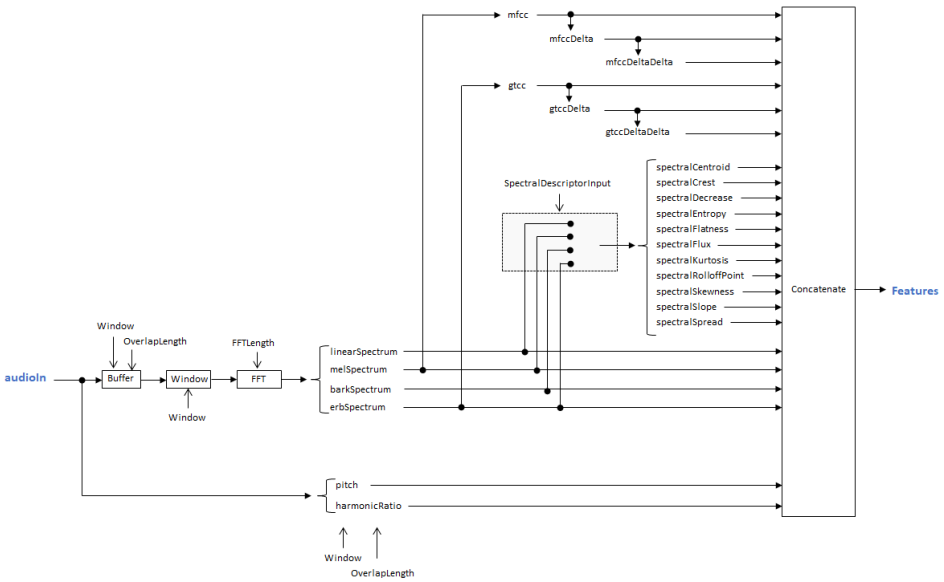
3.2.1 audioFeatureExtractor

Usarem la funció *audioFeatureExtractor.m* per a definir els atributs que analitzarem dels àudios, la qual ens permet de forma molt eficient parametritzar tot un corpus de fitxers continguts en una carpeta d'entrada. Com podem veure en la Figura 1 del diagrama de flux que la funció implementa, aquesta funció treballa amb atributs només de l'àmbit espectral o cepstral:

1. En primer lloc, una vegada aplicada la finestra i calculada la FFT de la trama (a on es defineixen paràmetres com son la longitud de la trama – *Window* – o el número de mostres de solapament entre trames – *OverlapLength*), es realitza el càlcul d'alguna **representació espectral** a partir d'aquesta. Per exemple, la opció *linearSpectrum* es queda només amb el mòdul de la primera meitat de la FFT corresponent a les freqüències positives (recordeu que l'espectre és simètric conjugat per a senyals d'àudio), a on podem definir també el marge de freqüències que ens quedem (per defecte de 0 a Fs/2, essent Fs la freqüència de mostratge). També es poden calcular altres representacions espectrals, totes elles basades en bancs de filtres basats en estudis sobre la percepció humana (*melSpectum*[3], *barkSpectrum*[4] i *erbSpectrum*[5]). En general, es poden també modificar o definir els paràmetres que per defecte tenen un valor associat per a cada tipus de representació (per exemple, per *melSpectrum* el rang freqüencial del banc de filtres, el tipus d'espectre – si retorna el mòdul o bé el mòdul quadràtic o potència –, el número de subbandes dins el rang freqüencial analitzat i el tipus de normalització de les amplituds dels filtres).
2. En segon lloc, una vegada definits els paràmetres bàsics de representació espectral, podem definir **altres paràmetres associats al tipus d'atributs d'àudio** a calcular. Si bé es poden usar les mateixes representacions espectrals com a atributs de sortida, en aquest punt podem definir com, a partir d'aquestes representacions espectrals, podem calcular altres atributs més específics i, sobretot, més compactes. En un primer rang tenim els atributs cepstrals (*mfcc* – Mel Frequency Cepstral Coeficients – o *gtcc* – GammaTone Cepstral Coeficients), a on els *gtcc* són una variant dels *mfcc* a on enlloc de partir de la representació amb l'espectre Mel partim de l'espectre Erb (a on la funció Gamma és la base de la resposta impulsional d'aquests tipus de filtres). A més, també podem calcular les velocitats i les acceleracions d'aquests dos conjunts de paràmetres,

[3] https://es.wikipedia.org/wiki/Escala_Mel
[4] https://ca.wikipedia.org/wiki/Escala_Bark
[5] https://en.wikipedia.org/wiki/Equivalent_rectangular_bandwidth

és a dir, la velocitat de canvi (*mfccDelta* o *gtccDelta*) i les acceleracions de canvi (*mfccDeltaDelta* o *gtccDeltaDelta*). En segon lloc, tenim altres atributs més específics obtinguts també a partir d'alguna de les 4 les representacions espectrals anteriors: *spectralCentroid*, *spectralCrest*, *spectralDecrease*, ..., *pitch*, *harmonicRatio*. Per alguns d'aquests, podem definir paràmetres de configuració, com per exemple el llindar entre 0 i 1 associat al càlcul del *spectralRolloffPoint*, o el tipus d'algoritme de càlcul per l'atribut de freqüència fonamental de la parla o *pitch*. En els atributs que conformen vectors de valors (com és el cas dels *mfcc* o *gtcc*) podem escollir el número de coeficients que volem calcular (per defecte 13).



3. Figura 1. Diagrama de flux de la funció audioFeatureExtractor.m

4. Una vegada generat l'objecte que retorna la funció *audioFeatureExtractor.m* (suposem que l'anomenem *aFE*) podem consultar, per exemple, la posició de cada paràmetre quan es faci el càlcul sobre un determinat senyal invocant *aFE.info*.

3.2.2 Extract Audio Features

Podeu usar la tasca *Extract Audio Features* per jugar a l'hora de generar les combinacions de paràmetres que escolliu per parametritzar els vostres àudios (el seu ús és opcional). És una interfície que ens permetrà fer proves i acabar de seleccionar tots els paràmetres de configuració d'una forma més visual, i finalment generar el codi Matlab per incrustar la crida a *audioFeatureExtractor.m* a la nostra funció final.

Per accedir a aquesta aneu a la pestanya EDITOR → New → Live Script i genereu una nova funció .mlx, i després seleccioneu dins la pestanya Task → Extract Audio Features aquesta tasca. Escolliu la configuració d'anàlisi que vulgueu provar, havent prèviament carregat algun senyal d'àudio de prova amb la seva freqüència de mostratge. Com es pot veure, en l'exemple de la Figura 2 són les variables *audio1* i *fs*, respectivament, i podem seleccionar tots els paràmetres de l'anàlisi que ens ofereix la funció. Si marquem la casella *Output summary* dins de l'opció final *Display results*, ens mostrarà a la dreta la organització dels P paràmetres de sortida d'una trama de senyal donada. La interfície ens generarà la matriu *features*, que conté les N trames de P paràmetres, de forma que podrem visualitzar, per exemple, els valors d'aquests paràmetres de totes les trames corresponent al senyal analitzat. A sota de tot de la interfície, si despleguem la

part inferior veurem el tros de codi Matlab que podrem incrustar per definir els paràmetres de crida de la funció *audioFeatureExtractor.m*.

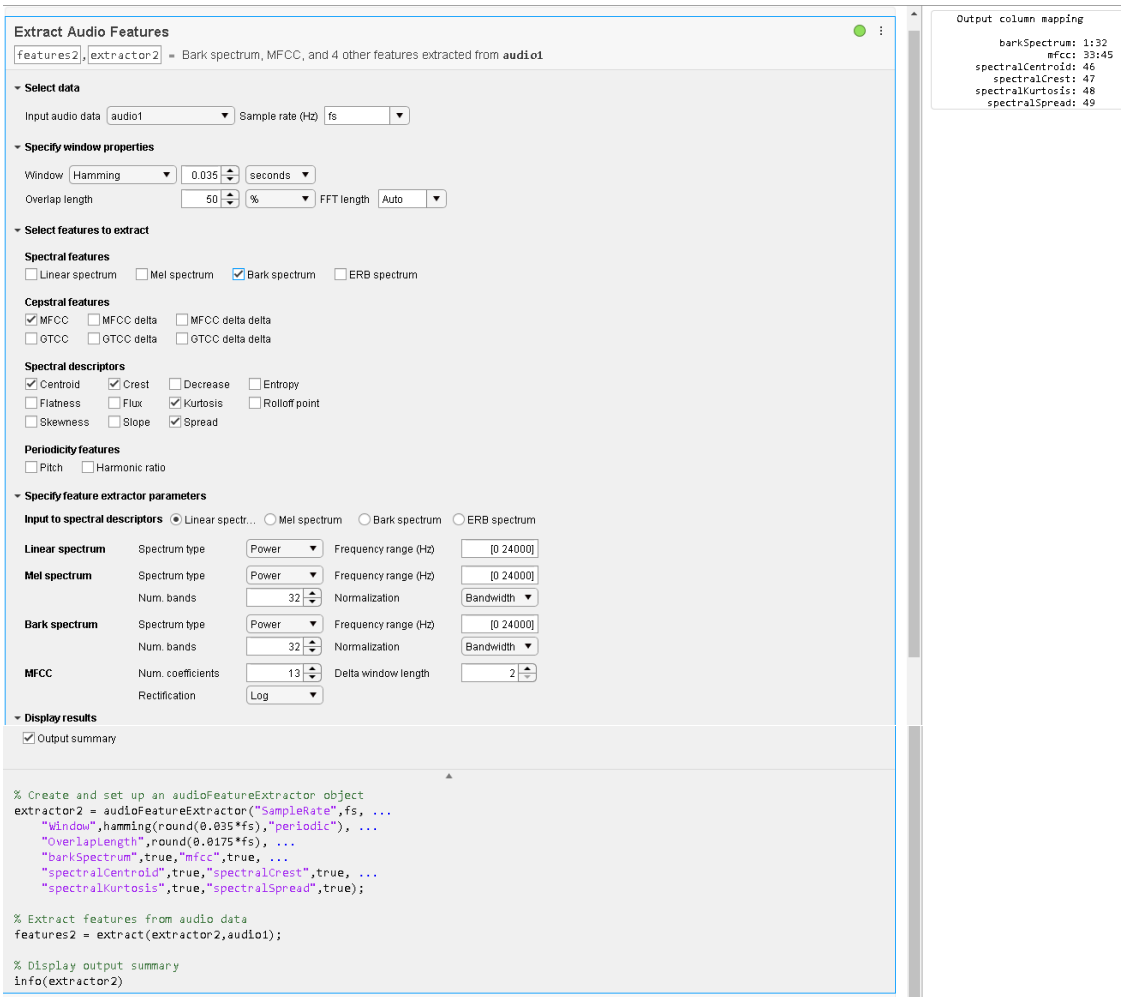


Figura 2. Exemple d'ús de la tasca Extract Audio Features dins del Live Editor del Matlab (fitxer amb extensió .mlx)

3.2.3 Extracció d'atributs de carpetes senceres

A l'hora de realitzar el càlcul dels atributs d'una carpeta sencera de fitxers ho farem amb crides a les funcions *audioDatastore.m*, *tall.m*, *cellfun.m*, *extract.m* i *gather.m*. Una vegada estiguin definits els diferents valors de les propietats d'entrada de la funció *audiofeatureExtractor*, i tenim la estructura que retorna aquesta funció en la variable *aFE*, si tenim la variable *adsTall* associada a una determinada base de dades (veure funció *audioDatastore.m* a l'apartat 3.1), fem les crides següents:

```
featuresTall = cellfun(@(x)extract(aFE,x),adsTall,"UniformOutput",false);
features = gather(featuresTall);
labels = ads.Labels;
```

que generarà en la variable *features* un cell array de N posicions (essent N el número de fitxers de la base de dades), a on en cada cel·la de l'array tindrem la matriu d'atributs del fitxer corresponent de mida M files i R columnes, essent M el número de finestres de senyal que caben en el fitxer i R el número d'atributs que hem configurat en la funció *audiofeatureExtractor.m*. A més, podrem accedir a les categories de cada trama amb la nova variable *labels*, que igual que *features*, serà un array categòric de longitud N amb les etiquetes de categoria dels N fitxers.

3.3 Funcions de classificació

Les funcions per entrenar i usar com a classificador les tenim disponibles dins la llibreria Statistical and Machine Learning Toolbox de Matlab. Quan usem les funcions d'entrenament dels classificadors usarem en aquestes crides també les dades pròpies per a realitzar aquest entrenament, mentre que quan usem les funcions de predicció o classificació de cada mètode usarem els conjunts de dades de test. Observeu que la majoria de funcions per avaluar els classificadors admeten dades en forma de matriu, és a dir, conjunts de vectors de test organitzats en files d'una matriu (cada fila representa un exemple a testejar, format per un vector de P components, essent P igual al nombre total d'atributs emprat i al nombre de columnes de la matriu de dades d'entrada a classificar).

A continuació donem algunes indicacions de les funcions que heu d'usar per a resoldre l'activitat pràctica. Tot i així, feu ús de l'eina *Help* que disposa el Matlab per a consultar els detalls concrets sobre dimensions i tipologies de les variables que cal passar als mètodes d'entrenament i de predicció o classificació. En les ajudes també veureu exemples que podeu executar abans d'implementar les crides, per verificar que enteneu tot el que es calcula i es necessita.

3.3.1 KNN (K-Nearest Neighbor)

L'entrenament del classificador KNN el podem realitzar amb la funció *fitcknn*, mentre que la funció per realitzar la classificació és la funció *predict*. La funció per entrenar es pot configurar amb la funció de distància que es vulgui amb el paràmetre 'Distance' (vegeu els valors possibles), i aquesta només pot prendre certs valors si la cerca es fa amb un arbre binari (vegeu a la variable 'NSMethod', la opció 'kd-tree') que permet accelerar el procés de cerca. A l'hora d'entrenar, també podem normalitzar les dades amb el paràmetre 'Standarize'. La funció de classificació pot retornar, a part de les etiquetes de les classes de cada exemple donat, les fiabilitats (*scores*) de cada exemple classificat.

3.3.2 CART (Classification and Regression Trees)

La funció per entrenar un classificador de tipus CART és *fitctree*, que té com a arguments principals el conjunt d'exemples d'entrenament (vectors d'atributs) i les respectives etiquetes de classe. La funció admet atributs tant reals com de tipus categòric (no és el nostre cas, a on tots els atributs son numèrics), i és tolerant a valors no calculats (p.ex. un vector que no té el valor d'un cert atribut). Podem usar la funció `view(treeModel,'Mode','graph')` per representar visualment el classificador, sempre i quan el número d'atributs no sigui massa gran. La variable *treeModel* es el model creat al executar *fitctree*. Una vegada s'ha entrenat el classificador CART, podem avaluar un conjunt de test amb la funció *predict*.

3.3.3 SVM (Support Vector Machines)

El classificador SVM és un classificador binari, és a dir, pensat per a resoldre problemes de classificació amb dues classes. Per tant, com el problema de classificació d'àudio pot ser, en general, un problema multi-classe, enlloc d'usar les funcions per entrenar SVM's usarem les que permeten entrenar compilacions de diversos SVM's binaris, amb una estratègia de binaritzar el problema de varies classes en diversos subproblemes de classificació binària. Per això, usarem la funció *fitcecoc*, a la qual li passarem igualment que en els altres casos les dades per entrenar el classificador i les etiquetes d'aquestes dades. A part, també li passarem el tipus de classificador binari, que en el nostre cas serà un SVM, i ho farem amb el paràmetre d'entrada 'Learner': `svmModel = fitcecoc(learnDB, learnLabels,'Learner',t)` (a on les variables *learnDB* i *learnLabels* contenen les dades i les etiquetes de classe, respectivament). En la variable `t` tindrem la configuració del classificador base, creat amb la funció templateSVM: `t =`

`templateSVM('KernelFunction','rbf','Standardize',true)` (exemple a on usem un Kernel de tipus radial, i a on li diem al classificador que normalitzi les dades restant a cada columna de la matriu de dades la mitja de cada paràmetre i dividint per la desviació estàndar). Per defecte, la funció usa un esquema de codificació basat en la estratègia de binarització One-vs-One (o sigui, cada classificador de la compilació es centra en distingir entre dues classes concretes, usant només les dades que té d'aquelles dues classes). Una vegada entrenat el classificador podem usar la funció *predict*, per a realitzar la classificació de nous exemples.

3.3.4 GMM (Gaussian Mixture Models)

Per entrenar un model de mescles de Gaussianes podeu usar la funció *fitgmdist*, a la qual li podeu passar el número de Gaussianes i les dades a usar per a l'entrenament. Recordeu que heu d'entrenar un model per a cada classe d'àudio que useu, de forma que després d'entrenar podem tenir accés a la probabilitat que un determinat model acústic (p.ex. de la classe "Bus") hagi generat una certa observació, i això ho podem obtenir amb la funció *pdf*, a la qual li passarem l'objecte del model. A aquesta funció també li podeu passar com a argument d'entrada una matriu de dades, és a dir, un conjunt de vectors de test, i us retornarà la probabilitat de cadascun dels vectors. Per a realitzar una classificació basada en el criteri ML (*Maximum Likelihood*), caldrà primer disposar de les probabilitats de cadascun dels models acústics de les classes que estem analitzant per als vectors de test, i en segon lloc podem usar la funció *max* de Matlab per a obtenir el model obté una probabilitat màxima per a cada vector. Us recomanem que useu bucles (*for*) i estructures de tipus cell array per emmagatzemar els models de cada classe (p.ex. `gmmModel{i} = fitgmdist(...)`).

En la crida a la funció *fitgmdist* veureu que heu d'indicar el nombre de components de la mescla (*k*), variable que haureu d'explorar en les vostres simulacions, és a dir, comprovar la eficàcia del classificador ML basat en GMMs per a diferents valors de *k*. En el cas de tenir problemes de convergència del model, proveu d'usar matrius de covariàncies diagonals (paràmetre 'CovarianceType'), o incloure el factor de regularització (paràmetre 'RegularizationValue'), etc. (vegeu els diferents paràmetres que permet la funció), i tingueu en compte que l'entrenament d'aquests models és molt sensible a la inicialització (podeu fer diversos entrenaments amb la opció 'Replicates', o modificar la forma de com iniciar els models amb la variable 'Start'). També tingueu en compte que l'entrenament pot ser sensible a les dades que li passem a la funció, per tant, depenent dels atributs d'àudio analitzats podem obtenir resultats força diferents.

4 Enunciat

L'objectiu de la pràctica és que demostreu amb una primera experiència l'ajust i validació d'un cert conjunt de tècniques tant de parametrització com de classificació d'àudio. A part de la funció *organize_dataset.m* de l'apartat 2 que organitza la base de dades per subcarpetes (una per cada categoria), haureu de generar una funció script que implementi el procés de càlcul d'atributs d'àudio de la vostra base de dades i el procés d'entrenament i test dels diferents mètodes de *Machine learning* estudiats usant un procés de validació creuada de 4 iteracions.

4.1 Funció general

Per aquest motiu, cal que dissenyeu una funció de tipus script en Matlab, amb nom *practica_audio.m* que inclogui els següents passos, a on el procés d'aprenentatge artificial es repeteixi 4 vegades usant un *4-fold cross-validation* (4FCV):

1. Definició de la base de dades a processar (funció *audioDatastore.m*), partició en 4 parts d'igual grandària (vegeu ús de la funció *splitEachLabel.m* a l'apartat 3.1) i

parametrització d'aquesta mitjançant els atributs d'àudio que definiu vosaltres (vegeu apartat 3.2). En aquest punt es recomana definir dos cell arrays de 4 posicions anomenats *features* i *labels* que continguin els cell arrays i vectors de categories de les 4 particions de dades, respectivament. En el cas dels atributs, aquest cell array de cell arrays podria inicialitzar-se com *features = cell(1,4);* per més tard definir el contingut de la primera cel·la com *features{1} = gather(features1_Tall);* essent *features1_Tall* el tall array d'atributs d'àudio associat a la primera partició de dades del 4-fold. Per saber el número de fitxers d'aquest fold ho podríem saber amb *length(features{1})*, i per referir-nos a la matriu d'atributs d'àudio del 4t fitxer d'aquesta partició ho faríem amb *features{1}{4}*. La etiqueta associada a aquest fitxer seria *labels{1}(4)*, a on la variable labels l'hem calculat assignant *labels{1} = ads1.Labels*, essent *ads1* l'estructura de fitxers de la primera partició que ens retorna la funció *splitEachLabel*.

2. Per a cadascuna de les 4 iteracions del 4FCV cal fer:
    - Calcular dos matrius (*learnDB* y *testDB*) que continguin els atributs de totes les trames del conjunt d'entrenament i de test d'aquella iteració, respectivament, així com també els corresponents vectors categòrics (*learnGT* y *testGT*) d'aquests dos conjunts. Per a la seva construcció haureu d'usar concatenació de matrius[6] (p.ex. per concatenar totes les matrius d'atributs dels fitxers associats al conjunt d'entrenament) i replicació d'etiquetes (p.ex. per copiar l'etiqueta associada a un determinat fitxer tants cops com finestres s'hagin obtingut en el procés de parametrització). Al final, el número de files de *learnDB* serà igual al número total de trames del conjunt d'entrenament (mentre que les columnes corresponen a número d'atributs), i coincidirà amb el número de components del vector categòric *learnGT* (tantes etiquetes com trames d'atributs de l'àudio), i de forma anàloga per *testDB* i *testGT*, respectivament. D'aquesta forma, tindrem les dades preparades per poder cridar a les funcions d'aprenentatge artificial, que esperen les dades amb aquest format.
    - Entrenament d'un mètode de classificació amb el subconjunt de dades d'entrenament (*learnDB* i *learnGT*). La idea és programar les crides per poder usar els 4 mètodes, però que en una determinada simulació escollirem un o altre, per poder fer diversos estudis.
    - Classificació del conjunt de test (*testDB*) amb el mètode de classificació, càlcul de la mètrica de validació comparant les etiquetes reals (*testGT*) amb les obtingudes amb el mètode de classificació (en aquest cas pot ser la taxa de reconeixement global o *Accuracy* ja que en principi la base de dades està força ben balancejada – totes les categories amb un nombre similar d'exemples). **Ajuda:** Siguin dos vectors A i B de la mateixa longitud amb les decisions del classificador (A) i amb les classes reals (B), essent aquests vectors categòrics, podem calcular les posicions a on coincideixen i a on difereixen fent: *okPositions = find(A==B);koPositions = find(A~=B);*

[6] Recordeu que per a concatenar vectors o matrius en Matlab ho podeu fer de forma senzilla. Per exemple, si volem inicialitzar una variable buida, i després concatenar-hi dos vectors columna de longitud 5 i 7, respectivament: M = [];M = [M;2*ones(5,1);3*ones(7,1)];

3. Mostrar els resultats d'eficiència de classificació (*Accuracy*) de cada iteració així com els amitjanats de les 4 iteracions per obtenir un resultat estadístic. Podeu mostrar els resultats per línia de comandes amb la funció `disp.m`.

Com es pot veure, el procés d'entrenament i validació seguirà una estratègia de validació creuada de 4 iteracions, és a dir, partiu el conjunt total de dades etiquetades disponibles en 4 subconjunts, i realitzeu 4 processos d'entrenament i test: en cadascun, un dels subconjunts és el de test, el qual és diferent per cada iteració del procés, i els 3 restants formen el conjunt d'entrenament.

La funció script *practica_audio.m* no ha de tenir capçalera de funció (no contindrà la comanda `function`), i a l'inici de la mateixa es definiran les següents variables de simulació:

- `path` (string): Adreça a la carpeta que conté la base de dades d'àudio, dins la qual hi ha les 15 subcarpetes de les categories amb els fitxers WAV a dins.
- `methodLM`: (string) Tipus de classificador (a escollir entre els valors 'knn','cart','svm','gmm').
- També definireu els paràmetres bàsics de cada classificador, seguint el conveni de posar primer el nom del classificador i després el nom del paràmetre (p.ex. `knn_K`, `svm_Kernel` o `gmm_N`).
- `percentOfDataSet` (valor numèric): definirà el % de dades que usem del corpus per a fer la simulació (vegeu funció *splitEachlabel*) per poder fer estudis més ràpids amb menys dades.

Per a generar diverses configuracions d'anàlisi només caldrà modificar les variables de simulació de l'inici de l'script Matlab, i així podreu generar diverses simulacions.

4.2 Estudis

Una vegada dissenyada la funció anterior (`practica_audio.m`) se us demana que realitzeu un mínim nombre d'estudis (al menys 4 o 5) que us permetin arribar a certes conclusions sobre configuracions de conjunts de paràmetres i classificadors usats que permeten arribar a millors taxes de classificació. Guardeu cada codi amb configuració concreta en un arxiu .m diferent (`practica_audio1.m`, `practica_audio2.m`, `practica_audio3.m`, `etc.`), ordenats segons la taxa de classificació obtinguda (`practica_audio1.m` serà el que us ha donat millors resultats d'*Accuracy*). Aquests estudis els reflectireu dins d'un document PDF (vegeu següent apartat) així com en les diverses versions de la funció que genereu per a cada estudi, modificant la configuració de variables de simulació. Què es demana que exploreu en aquests estudis?

- Modificar els atributs d'àudio escollits, o paràmetres que influeixen en el seu càlcul (p.ex. longitud finestra d'anàlisi). **Important**: Quan feu els estudis, teniu present l'ús de memòria segons les prestacions del vostre PC. Quants més atributs analitzeu, però sobretot, més trames per fitxer àudio tingueu (vigileu amb la durada de les trames i la seva separació en el càlcul dels atributs), més dades carregareu en memòria i pot passar que el PC passi a treballar de forma molt ineficient, fent molt accessos a disc. Intenteu tenir aquest aspecte present monitoritzant l'ús de memòria (l'ús de CPU serà intensiu quan es parametritzi el corpus i quan s'estigui entrenant o testejant amb un mètode d'aprenentatge artificial), i ajustant els paràmetres per tal d'evitar la saturació del vostre PC. Com ja hem comentat, la variable *percentOfDataSet* ha de ser una forma d'evitar

que el PC es saturi, i comenceu fent anàlisis més senzills amb 5-10% de les dades del corpus.

- Modificar el mètodes d'aprenentatge artificial escollit així com els seus paràmetres de configuració. Alguns paràmetres poden afectar també al rendiment del PC.

Els tres grups que assoleixin les 3 millors taxes de classificació global analitzant al menys el 50% de les dades del corpus tindran 1 (tercer millor), 2 (segon millor) i 3 punts (millor de tots) addicionals en la nota, podent així arribar a una nota màxima de 13 punts sobre 10. Aquesta competició serà vàlida només pel lliurament sobre 10 de la pràctica. Us animem a que durant les vostres proves, useu el fòrum de pràctiques per informar dels millors resultats que hagueu obtingut fins al moment, per incentivar la competició. Indiqueu al menys % de les dades usades, mètode ML usat i valor de l'Accuracy assolit però no doneu més detalls per fomentar que cada grup explori el màxim d'opcions per obtenir millors resultats que vosaltres. Els valors que assoliu es verificaran en el procés de correcció.

5 Lliurament

El lliurament de la pràctica es realitzarà amb un fitxer comprimit de tipus ZIP amb nom `practica_audio_X.zip`, essent X el número de grup, i contindrà:

1. El codi dissenyat (funció principal més possibles funcions que es cridin, fetes per vosaltres en Matlab) que segueixi els requisits marcats a l'apartat 4. S'inclouran tantes versions de la funció `practica_audio.m` com configuracions s'hagin testejat i reportat a l'informe en PDF, modificant els valors de les variables de simulació.
2. La memòria en format PDF, que inclourà el conjunt d'estudis demanats a l'apartat 4. Explicació dels resultats en funció dels paràmetres introduïts i les gràfiques o taules generades per Matlab en cada apartat. Incloeu també un estudi de la matriu de confusió[7] assolida per la millor configuració que heu obtingut.

El **lliurament de la pràctica sobre 10** es realitzarà al pou corresponent de la carpeta de l'assignatura i aquest **es tancarà el dijous 17 de Desembre a les 12h**. El lliurament posterior de la pràctica (data límit final: divendres 15 de Gener) implica límit màxim de qualificació igual a 7.

[7] https://ca.wikipedia.org/wiki/Matriu_de_confusi%C3%B3