SISTEMA DE RECONEIXEMENT D'EMOCIONS EN LA PARLA BASAT EN LA PARAMETRITZACIÓ DEL SENYAL DE VEU

Marcel González Poy

Grau de Tecnologies de la Telecomunicació

Aplicacions multimèdia basades en el processament del senyal

Consultor/a: **Lourdes Meler Corretjé**

Professor/a responsable de l'assignatura: **David García Solórzano**

15 de juny de 2016

Aquesta obra està subjecta a una llicència de Reconeixement-NoComercial-SenseObraDerivada 3.0 Espanya de Creative Commons

Fitxa del treball

Títol del treball:	Sistema de reconeixement d'emocions en la parla basat en la parametrització del senyal de veu
Nom de l'autor:	Marcel González Poy
Nom del consultor/a:	Lourdes Meler Corretjé
Nom del PRA:	David García Solórzano
Data de lliurament (mm/aaaa):	06/2016
Titulació o programa:	Grau de Tecnologies de la Telecomunicació
Àrea del Treball Final:	Aplicacions Multimèdia basades en processament del senyal
Idioma del treball:	Català
Paraules clau	*Speech Emotion Recognition*, *Feature extraction and selection, Classification Learner*

Resum

Aquest projecte consisteix en la implementació d'un sistema de reconeixement d'emocions en la parla. Es suggereix una arquitectura modular per obtenir més capacitat d'adaptació en el disseny, planificació i execució del projecte. Aquesta arquitectura es compon de dues etapes bàsiques: mòdul d'extracció de característiques i mòdul de classificació. El mòdul d'extracció és el responsable del processament del senyal d'entrada, dividint el treball en tres etapes: preprocessament, extracció de característiques i postprocessament. De la mateixa manera, hi ha dos procediments bàsics en el mòdul de classificació: entrenar el classificador per obtenir un model i, fer la predicció de l'emoció de la parla, classificant el senyal d'entrada d'acord amb el model. Les característiques extretes per a cada observació són el *pitch*, l'energia, la primera derivada de cada un i els 12 LFPC, amb el que s'obtenen 16 vectors de característiques. Per a cada un d'aquests, es calculen 12 paràmetres estadístics aconseguint, finalment, 192 valors per a cada observació. S'ha triat un classificador SVM multiclase, "un-contra-tots", amb el nucli de segon grau, degut al seu major rendiment. Tots els experiments s'han fet amb el corpus de veus emocionades: RAVDESS. S'ha assolit un 88% de precisió amb un conjunt de 4 emocions: enuig, alegria, tristesa, i neutral. Tots els processos s'han representat matemàticament i s'han implementat amb el programari MATLAB.

Abstract

This project consist in the implementation of a SER: Speech Emotion Recognition system. A modular architecture is suggested to get more adaptability in the design, planning and execution of the project. This architecture is composed of two basic stages: feature extraction module and classification module. The extraction module is the responsible of the input signal processing, dividing the work into three steps: pre-processing, features extraction and post-processing. In the same way, there are two basic procedures in the classification module: train the classifier to get a model and, make the prediction of the speech emotion, classifying the input signal according to the model. The features extracted for every observation are pitch, energy, first derivative of each one and the twelve LFPC, so 16 features vectors are obtained. For each of these, 12 statistical parameters are computed getting,

finally, 192 values for each observation. A multi-class,“one-vs-all”, SVM classifier, with quadratic kernel, has chosen because of its greater performance. All the experiments have done with RAVDESS, the corpus of emotional voices. An 88 % of accuracy has been reached with a set of 4 emotion: angry, happy, sad, and neutral. All the processes have represented mathematically and have implemented with MATLAB software.

Resumen

Este proyecto consiste en la implementación de un sistema de reconocimiento de emociones en el habla. Se sugiere una arquitectura modular para obtener más capacidad de adaptación en el diseño, planificación y ejecución del proyecto. Esta arquitectura se compone de dos etapas básicas: módulo de extracción de características y módulo de clasificación. El módulo de extracción es el responsable del procesamiento de la señal de entrada, dividiendo el trabajo en tres etapas: preprocesamiento, extracción de características y postprocesamiento. De la misma manera, existen dos procedimientos básicos en el módulo de clasificación: entrenar el clasificador para obtener un modelo y, hacer la predicción de la emoción del habla, clasificando la señal de entrada de acuerdo con el modelo. Las características extraídas para cada observación son el *pitch*, la energía, la primera derivada de cada uno y los 12 LFPC, obteniendo un total de 16 vectores de características. Para cada uno de estos, se calculan 12 parámetros estadísticos consiguiendo, finalmente, 192 valores para cada observación. Se ha elegido un clasificador SVM multiclase, “uno-contra-todos”, con el núcleo de segundo grado, debido a su mayor rendimiento. Todos los experimentos se han hecho con el corpus de voces emocionadas: RAVDESS. Se ha alcanzado un 88% de precisión con un conjunto de 4 emociones: enojado, feliz, triste, y neutral. Todos los procesos se han representado matemáticamente y se han implementado con el software MATLAB.

Al meu pare

Síntesi

En aquest TFG s'ha realitzat el disseny i implementació d'un sistema que permet reconèixer l'emoció subjacent en una frase prèviament enregistrada. El disseny proposat sorgeix de l'estudi de treballs similars i la recerca d'informació en les disciplines que intervenen en la creació de sistemes SER com: el processament del senyal digital de la veu, la caracterització acústica de la veu, reconeixement de patrons, estadística o màquines d'aprenentatge automàtic. La solució de programari s'ha desenvolupat íntegrament amb MATLAB. S'han creat un conjunt de funcions que permeten l'execució del sistema des de la línia de comandes i una interfície gràfica que permet un ús més intuïtiu i transparent per a l'usuari.

El sistema consta de dos blocs principals: el mòdul d'extracció de característiques i el mòdul de classificació. L'execució del sistema s'efectua en dos fases: la fase d'entrenament i la fase de predicció. En la fase d'entrenament el sistema utilitza els dos mòduls per crear un model que servirà per a les posteriors prediccions. En la segona fase, es fa ús del mòdul d'extracció i del model creat per a predir l'emoció de la frase. Tenint en compte aquest disseny, s'ha fet una implementació modular que atorga flexibilitat i abstracció entre els diferents mòduls, submòduls i processos del sistema.

En el mòdul d'extracció de característiques estan compresos els tres submòduls que realitzen les tasques de processament del senyal: preprocessament, extracció de característiques i postprocessament. El preprocessament s'ha dividit en les següents funcions: filtre passa alt, normalització energètica, remostreig, detecció d'inici i final de la frase i, segmentació i enfinestrament. El submòdul d'extracció es el responsable de la parametrització de la informació acústica del senyal. Les característiques que s'han escollit són: el *pitch*, l'energia, la primera derivada de cada un d'aquests (que permet tenir una visió de la variació a nivell temporal) i 12 coeficients LFPC (*Log Frequency Power Coefficients*). De cada una, s'han calculat 12 paràmetres estadístics: *mean, min, max, standard deviation*, *skewness, kurtosis*, *1st quartile, 2nd quartile, 3rd quartile, interquartile range, semi-interquartile deviation* i *90 percentile*. Finalment, en el postprocessament s'ha implementat la selecció de característiques basada en filtres amb el mètode d'informació mútua.

S'ha escollit el sistema de classificació SVM (*Support Vector Machines*). Aquest, s'ha implementat, juntament amb la validació creuada *k-fold cross validation* en el mòdul de classificació.

El sistema permet introduir manualment els paràmetres opcionals: freqüència de mostreig, temps de finestra i freqüència de tall del filtre passa alt. També permet l'entrenament de classificadors amb diferents conjunts d'emocions i bases de dades.

Les proves realitzades han donat com a resultat models amb una precisió de fins al 88% amb una freqüència de mostreig de 22050 Hz, un temps de finestra de 0.023 segons i una freqüència de tall de 100 Hz, per a un conjunt de dades amb 4 emocions: enuig, alegria, tristesa i neutral. Aquest conjunt d'emocions es l'utilitzat en treballs similars d'aquesta bibliografia. Totes les proves s'ha dut a terme amb la base de dades RAVDESS.

Agraïments

Vull donar les gràcies al professor responsable de l'assignatura, David García Solórzano, per aconsellar-me i ajudar-me en l'elecció del projecte. També, vull donar les gràcies a la consultora, Lourdes Meler Corretjé, pel seguiment realitzat durant aquest TFG.

Agrair a la meva mare i germans el suport durant aquesta etapa.

Agrair al meu fill, Gerard, que tot i privar-me de dormir moltes nits i no ser-ne conscient (encara), m'ha aportat la motivació i voluntat per seguir endavant.

Finalment, vull donar el meu agraïment més espacial a Mònica i Eleonor per estar sempre al meu costat i donar-me tot el que tenien a l'abast per a poder superar aquest projecte i gran part de la carrera.

Índex

Fitxa del treball **III**

Síntesi **VI**

Agraïments **VIII**

Índex **IX**

Índex de figures **XI**

Índex de taules **XII**

Capítol 1 Introducció **1**

1.1 Context i justificació del treball....1

1.2 Objectius....3

1.3 Enfocament i metodologia....3

1.4 Planificació....4

1.5 Sumari de productes obtinguts....8

1.6 Resum de la resta de capítols....8

Capítol 2 Estat de l'art dels sistemes SER **10**

2.1 Les emocions i la parla....10

2.2 Acústica fonètica....11

2.2.1 Model lineal de la producció de la veu....11

2.2.2 Extracció de característiques del senyal de veu....12

2.3 Sistemes de classificació....18

2.3.1 Support Vector Machines (SVM)....19

2.3.2 Hidden Markov Models (HMM)....20

2.3.3 k-Nearest Neightbors (k-NN)....21

2.4 Obtenció de dades....22

Capítol 3 Disseny **24**

Capítol 4 Fases d'execució del sistema **27**

4.1 Fase d'entrenament....27

4.1.1 Càrrega de dades al sistema....28

4.2 Fase de predicció....29

Capítol 5 Implementació: Mòdul d'extracció de característiques **30**

5.1 Preprocessament....30
5.1.1 Filtratge....31
5.1.2 Normalització....32
5.1.3 Remostreig....32
5.1.4 End Point Detect....33
5.1.5 Segmentació i enfinestrament....34
5.2 Extracció de característiques....36
5.2.1 Pitch....37
5.2.2 Energia....39
5.2.3 Short time Log Frecuency Power Coefficients (LFPC)....39
5.2.4 Paràmetres estadístics....43
5.3 Selecció de característiques (postprocessament)....46
5.3.1 Informació mútua....46
Capítol 6 Implementació: Mòdul de classificació 48
6.1 Sistema de classificació SVM....49
6.2 Validació creuada....50
Capítol 7 Interfície gràfica: CTSSER GUI 52
7.1 Implementació....52
7.2 Ús de la interfície gràfica....53
Capítol 8 Proves i resultats 56
8.1 Prova 1....56
8.2 Prova 2....59
8.3 Prova 3....60
Capítol 9 Conclusions 61
9.1 Acompliment dels objectius....61
9.2 Conclusions....62
9.3 Línies de treball futur....64
Bibliografia 66
Annex 1 Codi font 68

Índex de figures

Diagrama de Gantt del TFG....4
Model de producció de la veu [7]....12
Categories de característiques de la veu [10]....14
Esquema d'extracció de característiques LFPC [13]....15
Segmentació i enfinestrat del senyal [9]....17
Comparativa entre mètodes de classificació [7]....19
Exemple de classificació amb SVM....20
Exemple de classificació amb HMM....20
Exemple d'estats d'un sistema HMM....21
Diagrama de blocs del projecte....24
Diagrama de la fase d'entrenament del sistema....27
Diagrama de la fase de predicció del sistema....29
Resposta del filtre passa alt....31
Espectrograma d'un àudio de la base de dades RAVDESS....32
Funcionament endPointDetect....34
Finestra Hamming....36
Distribució de densitat: pitch....38
Distribució de densitat: derivada del pitch....38
Distribució de densitat: log energy....39
Distribució densitat: derivada log energy....39
Distribució espectral d'energia....42
Distribució de densitat: LFPC....43
Interfície gràfica CTSSER GUI....53
Matriu de confusió (22050Hz -23 ms)....58
Matriu de confusió (48000Hz - 21 ms)....58
Matriu de confusió amb 7 emocions....60

Índex de taules

Tasques del TFG.....5
Temps de finestra recomanats en funció de la freqüència de mostreig.....35
Banc de filtres LFPC.....41
Paràmetres estadístics per ordre de rellevància.....47
Precisió del classificador per a una freqüència de mostreig de 16000 Hz.....56
Precisió del classificador per a una freqüència de mostreig de 22050 Hz.....57
Precisió del classificador per a una freqüència de mostreig de 44100 Hz.....57
Precisió del classificador per a una freqüència de mostreig de 48000 Hz.....57
Precisió en funció del filtre HPF.....59

Capítol 1 Introducció

El Treball Final de Grau (TFG) ha de servir per posar en pràctica les competències adquirides durant el Grau. És, per tant, la síntesi de molts anys d'estudi i permet assimilar molts dels coneixements previs a través de la recerca i la implementació pràctica del projecte.

Aquest TFG, emmarcat dins l'àrea del processament del senyal, consisteix en la realització d'un sistema de reconeixement d'emocions en la parla. En conseqüència, hi ha un component important de recerca en els camps del processament digital del senyal de veu, reconeixement de patrons i màquines d'aprenentatge automàtic. A més, s'han consolidat conceptes d'assignatures com senyals i sistemes I i II i, s'han posat en pràctica les aptituds en programació. També, s'han entrenat competències transversals com la redacció de textos cientificotècnics, la metodologia de recerca o la planificació i execució de projectes.

1.1 Context i justificació del treball

La comunicació verbal és el mètode d'interacció més generalitzat entre les persones. Per tant, es natural pensar que ho ha de ser en la interacció entre «humà-màquina» o «humà-ordinador». És per aquest motiu que el reconeixement de la parla i la síntesi de veu són camps en constant desenvolupament; a mesura que hi ha avenços en aquest àmbit sorgeixen noves aplicacions de gran utilitat en diversos camps com la industria de l'entreteniment i consum, la medicina, la seguretat o l'educació entre d'altres.

Ara bé, cal pensar que moltes d'aquestes aplicacions es basen en la informació lingüística de la parla, és a dir «allò que es diu», i no pas en la informació paralingüística, «com es diu» [1]. En aquest últim punt és on es pot millorar aquesta tecnologia per tal d'oferir sistemes d'interacció molt més naturals i sense ambigüitats. La

llista de propietats paralingüístiques és molt llarga: gènere, edat, qualitat de la veu, patologies, emocions, entre moltes altres. D'aquestes, l'emoció juga un paper molt important en diversos camps, com la psicologia, la seguretat o la síntesi de veu. Això a obert un nou camp d'investigació: el reconeixement d'emocions en la parla (SER, *Speech Emotion Recognition*) [2].

Així doncs, calen sistemes de reconeixement d'emocions en la parla que ofereixin aquest grau de naturalitat en sistemes de reconeixement de la parla o, fent el procediment invers, aportin naturalitat i menys ambigüitat als sistemes de síntesi de veu. El reconeixement d'emocions en la veu també pot resultar una eina de gran utilitat en el camp de la medicina i la psicologia. Saber l'estat d'ànim d'un pacient en una teràpia pot ajudar en la diagnosi del pacient. En general, l'aportació més important d'aquesta nova tecnologia és la gran quantitat d'aplicacions que se'n poden derivar, tot i que encara s'han de realitzar sistemes robustos i amb més precisió.

En aquest TFG s'implementa un sistema SER. Per fer-ho, s'ha de tenir en compte com afecten les emocions en la parla. S'ha de tenir accés a bases de dades (o crear-les) amb diferents mostres de veu de diferents oradors amb estats emocionals coneguts. Aquestes dades s'han de processar de manera que se'n pugui extreure les característiques desitjades i, després, classificar-les i crear un model que servirà per fer les prediccions de les veus amb emocions desconegudes. Finalment s'ha de validar l'experiment i contrastar els resultats.

Alguns sistemes SER estudiats presenten certs impediments, com és el fet de estar condicionats a l'idioma de la base de dades emprada o el nombre d'emocions (classes) a a analitzar. Per tal de superar aquest impediment, aquest sistema SER s'ha dissenyat de manera que permeti utilitzar qualsevol base de dades i nombre de classes. Així, aquest projecte tracta de crear un sistema d'entrenament de classificadors per a la creació de models que serviran per al reconeixement d'emocions. Per tant, s'ha de veure més com una eina que com una aplicació final.

1.2 Objectius

Existixen dos objectius primordials en la realització d'aquest TFG, que s'enumeren a continuació:

- Implementar una solució de programari acceptable per al reconeixement d'emocions en la parla. És a dir, realitzar un programa amb MATLAB que permeti extreure determinades característiques d'una mostra de veu donada i reconèixer, gracies a aquestes, l'estat emocional de l'orador. Com a objectiu secundari, cal esmentar la possibilitat de dissenyar i implementar una interfície gràfica amb MATLAB.
- Adquirir una base sòlida de coneixements en l'àmbit del processament d'àudio i en concret del senyal de veu. Conèixer els principals mètodes per a l'extracció de característiques i els sistemes de classificació, tant a nivell pràctic com a nivell teòric.

1.3 Enfocament i metodologia

Per tal de dur a terme el desenvolupament de forma exitosa s'ha de conèixer la teoria subjacent. Ara bé, l'abast del projecte en quant a continguts és força exigent amb relació al temps per realitzar-lo. En conseqüència, s'ha optat per una estratègia que permeti adquirir les competències enumerades als objectius i alhora crear el producte desitjat, seguint les indicacions dels projectes similars de la bibliografia.

El primer pas ha estat l'estudi del funcionament i realització dels sistemes SER, que de manera general segueix la següent metodologia:

- Creació o cerca de la base de dades amb veus emocionades.
- Processament del senyal (extracció de característiques).
- Selecció de característiques i classificació.

Aquest primer estudi dona una visió global que permet delimitar els elements que s'utilitzen per a la implementació. D'aquesta manera s'acota l'abast i s'aprofundeix, exclusivament, en els conceptes importants per a aquest projecte.

El següent pas ha estat el desenvolupament, durant el qual s'assimilen els conceptes teòrics i es posen en pràctica:

- Estudi del programari utilitzat (MATLAB).
- Disseny modular.
- Implementació per blocs.
- Assajos i proves.

Per altra banda, seguint el procediment de la UOC en quant a les proves d'avaluació continuada (PAC), cada una d'aquestes parts es documenta i serveix com a lliurable per al seguiment del TFG. Aquesta documentació s'utilitza com a base per a la redacció de la memòria del projecte.

1.4 Planificació

El següent diagrama de Gantt mostra la planificació del projecte:

Id	Nombre de tarea	Duración	Comienzo	Fin
1	**1 PAC1: Definició**	**12 días**	**jue 25/02/16**	**mié 09/03/16**
2	1.1 Definició del TFG	9 días	jue 25/02/16	sáb 05/03/16
3	1.2 Documentació de la proposta	3 días	lun 07/03/16	mié 09/03/16
4	1.3 Lliurament PAC1	0 días	mié 09/03/16	mié 09/03/16
5	**2 PAC2: Estat de l'art**	**18 días**	**jue 10/03/16**	**mié 30/03/16**
6	2.1 Cerca de recuresos	8 días	jue 10/03/16	vie 18/03/16
7	2.2 Anàlisi dels recursos	5 días	sáb 19/03/16	jue 24/03/16
8	2.3 Documentació de l'estat	5 días	vie 25/03/16	mié 30/03/16
9	2.4 Lliurament PAC2	0 días	mié 30/03/16	mié 30/03/16
10	**3 PAC3: dissney i implementació**	**48 días**	**jue 31/03/16**	**mié 25/05/16**
11	3.1 Disseny	5 días	jue 31/03/16	mar 05/04/16
12	**3.2 Implementació mòdul extracci**	**18 días**	**mié 06/04/16**	**mar 26/04/16**
13	3.2.1 Preprocessament	6 días	mié 06/04/16	mar 12/04/16
14	3.2.2 Extracció de característiqu	6 días	mié 13/04/16	mar 19/04/16
15	3.2.3 Postprocessament	6 días	mié 20/04/16	mar 26/04/16
16	**3.3 Implementació mòdul calssific**	**10 días**	**mié 27/04/16**	**sáb 07/05/16**
17	3.3.1 sistema de classificació	6 días	mié 27/04/16	mar 03/05/16
18	3.3.2 validació creuada	4 días	mié 04/05/16	sáb 07/05/16
19	3.4 Implementació GUI	6 días	lun 09/05/16	sáb 14/05/16
20	3.5 Documentació de la implement	9 días	lun 16/05/16	mié 25/05/16
21	3.6 Lliurament PAC3	0 días	mié 25/05/16	mié 25/05/16
22	**4 PAC4: Memòria**	**18 días**	**jue 26/05/16**	**mié 15/06/16**
23	4.1 Documentació memòria	18 días	jue 26/05/16	mié 15/06/16
24	4.2 Lliurament PAC4	0 días	mié 15/06/16	mié 15/06/16
25	**5 PAC5A: Presentació**	**12 días**	**jue 16/06/16**	**mié 29/06/16**
26	5.1 Elaboració de la presentació	12 días	jue 16/06/16	mié 29/06/16
27	5.2 Lliurament PAC5A	0 días	mié 29/06/16	mié 29/06/16
28	**6 PAC5B: Defensa**	**4 días**	**jue 30/06/16**	**lun 04/07/16**
29	6.1 Defensa	4 días	jue 30/06/16	lun 04/07/16

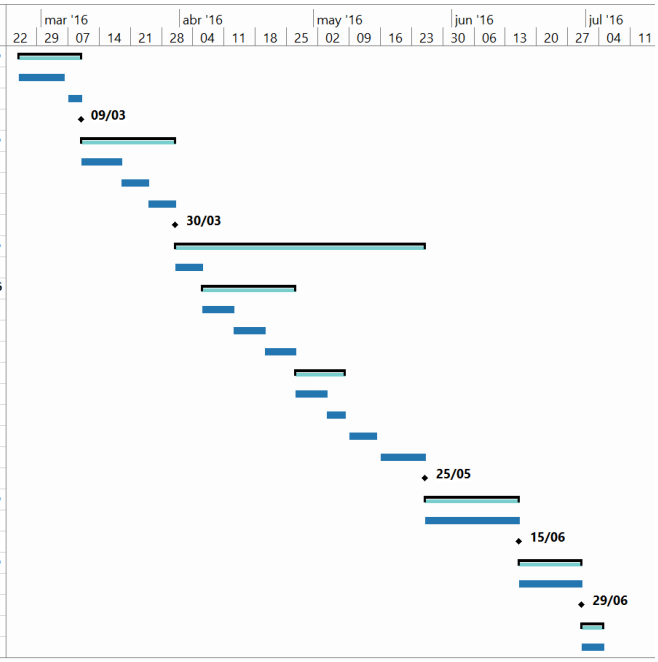

Figura 1: Diagrama de Gantt del TFG

El diagrama segueix l'esquema de les proves d'avaluació continuada (PAC), marcades com a fites. Per a cada fita es realitzen diferents tasques que s'enumeren en la taula 1 i s'expliquen resumidament a continuació.

Taula 1: Tasques del TFG

Codi	Tasca	Duració	Data inici	Data Fi
1.1	Definició del TFG	9 dies	25/02/16	05/03/16
1.2	Documentació de la proposta	3 dies	07/03/16	09/03/16
2.1	Cerca de recursos	8 dies	10/03/16	18/03/16
2.2	Anàlisi dels recursos	5 dies	19/03/16	24/03/16
2.3	Documentació de l'estat	5 dies	25/03/16	30/03/16
3.1	Disseny	5 dies	31/03/16	05/04/16
3.2.1	Preprocessament	6 dies	06/04/16	12/04/16
3.2.2	Extracció de característiques	6 dies	13/04/16	19/04/16
3.3.3	Postprocessament	6 dies	20/04/16	26/04/16
3.3.1	sistema de classificació	6 dies	27/04/16	03/05/16
3.3.2	validació creuada	4 dies	04/05/16	07/05/16
3.4	Implementació GUI	6 dies	09/05/16	14/05/16
3.5	Documentació de la implementació	9 dies	16/05/16	25/05/16
4.1	Documentació memòria	18 dies	26/05/16	15/06/16
5.1	Elaboració de la presentació	12 dies	16/06/16	29/06/16
6.1	Defensa	4 dies	30/06/16	04/07/16

1.1	**Definició del TFG**
Durada	9 dies
Descripció	Definició el projecte, motivació i definició de l'abast i cerca de bibliografia

1.2	**Documentació de la proposta**
Durada	3 dies
Descripció	Documentació de la definició del TFG per al lliurament de la PAC1

2.1	Cerca de recursos
Durada	8 dies
Descripció	Cerca de fonts d'informació per a l'estudi previ i desenvolupament posterior. Inclou la cerca de bases de dades amb veus emocionades.

2.2	Anàlisi dels recursos
Durada	5 dies
Descripció	Triatge i estudi dels recursos importants per al projecte. Aquests recursos conformen al bibliografia final.

2.3	Documentació de l'estat
Durada	5 dies
Descripció	Documentació de l'estat de l'art del TFG per al lliurament de la PAC2

3.1	Disseny
Durada	5 dies
Descripció	Elaboració d'un esquema modular de funcionament del sistema que permet la implementació per fases. Inclou la definició final del sistema i l'elecció dels elements que el conformen.

3.2.1	Preprocessament
Durada	6 dies
Descripció	Estudi i implementació de cada funció del submòdul de preprocessament del senyal: filtres, mostreig, segmentació i enfinestrament, detecció de silencis. Inclou la gestió de càrrega de dades al sistema.

3.2.2	Extracció de característiques
Durada	6 dies
Descripció	Estudi de les característiques escollides i implementació de les funcions que permeten extreure-les del senyal.

3.3.3	Postprocessament
Durada	6 dies
Descripció	Estudi del mètode de selecció de característiques escollit i posterior implementació.

3.3.1	sistema de classificació
Durada	6 dies
Descripció	Estudi i implementació del sistema de classificació

3.3.2	validació creuada
Durada	4 dies
Descripció	Estudi i implementació de la funció de validació del sistema

3.4	Implementació GUI
Durada	6 dies
Descripció	Disseny de la interfície gràfica. Implementació i proves de funcionament.

3.5	Documentació de la implementació
Durada	9 dies
Descripció	Documentació de cada pas de la implementació, demostrant el correcte funcionament de cada part. Inclou la realització d'experiments i la documentació dels resultats. Aquest document representa el lliurable de la PAC3.

4.1	Documentació memòria
Durada	18 dies
Descripció	Elaboració de la memòria que representa el lliurable de la PAC4.

5.1	Elaboració de la presentació
Durada	12 dies
Descripció	Creació d'una presentació virtual: presentació en *powerpoint* o similar, amb narració de fons i vídeo tutorial de la GUI. Aquest és el lliurable de la PAC5A.

6.1	Defensa
Durada	4 dies
Descripció	Preparació i estudi del projecte. Assimilació dels coneixements adquirits i preparació de les respostes a les preguntes formulades pel tribunal; lliurament de la PAC5B.

1.5 Sumari de productes obtinguts

Els productes obtinguts amb aquest TFG són:

1. **Solució de programari**

   Un conjunt de funcions i una GUI desenvolupades amb MATLAB.

2. **Memòria del projecte**

   Documentació del projecte. Inclou la metodologia i procediments seguits per a l'acompliment dels objectius d'aquest treball, les conclusions i línies de treball futur.

3. **Presentació virtual**

   Presentació esquemàtica dels trets principals del projecte.

4. **Vídeo demostratiu de la interfície gràfica**

   Demostració del funcionament de la GUI.

1.6 Resum de la resta de capítols

Els capítols d'aquest document s'estructuren seguint l'ordre de la planificació. Així, el capítol 2 fa referència a l'estat de l'art. Els capítols 3, 4, 5, 6 i 7 comprenen el disseny i implementació. En el capítol 8 es detallen els diferents experiments i, finalment, en el capítol 9 s'enumeren les conclusions i línies de treball futur. A continuació es resumeixen breument els aspectes més importants de cada capítol.

En el capítol 2, «Estat de l'art dels sistemes SER», es fa un estudi dels sistemes de reconeixement d'emocions en la parla. S'explica la metodologia emprada en projectes similars i s'analitzen les diferents eines existents per dur a terme la implementació. S'ha fet un recull d'informació de diferents camps com: acústica fonètica, processament digital del senyal de veu, extracció i selecció de característiques i sistemes de classificació. També inclou un apartat amb la informació necessària per a l'obtenció de dades o, en el seu defecte, l'elecció d'una base de dades ja existent.

En el capítol 3, «Disseny», es presenta el disseny proposat amb un diagrama de blocs i les funcions que es realitzen a cada un dels dos blocs principals: el mòdul d'extracció de característiques i el mòdul de classificació. Aquest disseny permet executar el sistema en dues fases: fase d'entrenament i fase de predicció. S'explica la seva implementació en el capítol 4, «Fases d'execució del sistema».

La implementació del sistema s'ha dividit en tres capítols. En el primer, el capítol 5 «Implementació: Mòdul d'extracció de característiques», s'expliquen cada un dels submòduls i els processos que conformen el mòdul d'extracció: preprocessament, extracció i postprocessament. De la mateixa manera, en el capítol 6 «Implementació: Mòdul de classificació», s'enumeren tots els procediments seguits per a posar en pràctica el mòdul de classificació. Finalment, en el capítol 7 «Interfície gràfica: CTSSER GUI», s'explica la implementació de la GUI i el seu funcionament.

En el capítol 8, «Proves i resultats», s'analitzen els resultats de 3 experiments. En el primer, s'avaluen els resultats en funció de la freqüència de mostreig i els temps de finestra recomanats. En el segon, es mesura l'efecte del filtre en la precisió dels classificadors. El darrer experiment serveix per posar a prova la versatilitat del sistema respecte a l'anàlisi de conjunts d'emocions diversos.

En l'últim capítol, «Conclusions», es fa una valoració del projecte en vers als objectius aconseguits i les conclusions extretes. A més, s'enumeren possibles línies de treball futur tant a nivell teòric com pràctic.

També s'afegeix un annex , «Codi font», amb cada una de les funcions implementades per a aquest projecte.

Capítol 2 Estat de l'art dels sistemes SER

L'estudi del reconeixement d'emocions en la parla és un camp d'investigació interdisciplinari. Hi ha molts experts de diferents branques de la ciència treballant en el desenvolupament de sistemes SER; matemàtics, enginyers, metges o psicòlegs entre d'altres experts de la veu i la parla. La raó és evident: per entendre un sistema SER i poder-lo desenvolupar correctament, s'ha d'aprofundir en el coneixement de les emocions i com aquestes poden afectar a l'aparell fonador; també, s'ha d'estudiar com aquestes alteracions físiques - de l'aparell fonador - afecten a la producció de la veu i, per tant, de la parla; cal conèixer les característiques acústiques de la veu i com es poden capturar i emmagatzemar les mostres d'una manera adequada per tal de fer un processat d'aquest senyal que permeti extreure les característiques per separat, seleccionar-les i associar-les a determinats estats emocionals.

En els següents apartats es fa un resum del estat de l'art en cada una d'aquestes disciplines que conformen els sistemes SER, fent referència a diferents estudis i autors.

2.1 Les emocions i la parla

Intuïtivament, es pot entendre que les emocions afecten físicament el nostre cos; canvis en el ritme cardíac, en la pressió o en la tensió muscular són els més evidents, tot que n'hi ha molts més òbviament. Aquestes alteracions afecten la producció de la parla d'una forma determinada, amb la qual cosa, si es parametritza la veu, se'n poden extreure les característiques subjacents provocades per l'efecte de cada emoció en l'aparell fonador.

Tot i que per a projectes futurs s'hauria d'aprofundir en l'estudi de les emocions i com aquestes són transmeses a través de la parla, ja que és bàsic per a un complert enteniment de la matèria, no és la intenció d'aquest projecte endinsar-se en el

coneixement psicològic i fisiològic. Així doncs, de cara a determinar quines emocions seran utilitzades en aquest projecte, es fa un recull d'informació d'altres treballs d'aquest tipus.

En tots els articles i treballs d'aquesta bibliografia s'evidencia que a dia d'avui no hi ha un consens de quines són les emocions que es poden transmetre a través de la parla. En l'anàlisi de Ververidis i Kotropoulos [3], es representen un total de 29 emocions en 64 bases de dades de mostres de veu; tot i que en la majoria d'aquestes el nombre d'emocions que es descriuen varia entre 4 i 6. Altres estudis [4][5][6], proposen fins a 15 emocions detectables en la parla, tot i que evidencien el fet de que n'hi ha de primordials. En concret, les més freqüents i conegudes són les següents: enuig, alegria, tristesa, por, sorpresa, disgust i neutral. En canvi, en altres treballs [7][8] s'estudien 4 i 3 estats anímics respectivament. Per tant, es pot concloure que el nombre d'estats escollit dependrà de l'aplicació o finalitat.

2.2 Acústica fonètica

Com s'ha comentat anteriorment, les emocions afecten a la veu a causa dels processos psicosomàtics. Cal doncs, estudiar la fisiologia de l'aparell fonador i representar-lo en un model simple que descrigui el seu comportament i permeti parametritzar el senyal de veu de tal manera que se'n pugui extreure certes característiques per a la seva posterior anàlisis. Hi ha estudis molt extensos sobre la fisiologia de la fonació [9]. En aquest apartat se'n remarquen alguns punts d'interès per al propòsit d'aquest TFG.

2.2.1 Model lineal de la producció de la veu

Per tal de parametritzar la veu s'ha creat un model anomenat «model lineal» basat en el teoria font-filtre de Fant (1960). Aquesta teoria, en el seu intent de simplificació, considera només tres elements en la producció de la veu: l'excitació (el flux glòtic modulat per la vibració de les cordes vocals), la transmissió (condicionada per la configuració i la ressonància del tracte vocal supraglòtic: cavitats faríngia i oral) i la radiació (deguda a la configuració de l'obertura de la boca per la posició dels llavis). El model assumeix el comportament lineal del sistema i la no variació en el temps del tracte vocal. Aquestes dues premisses permeten considerar de forma independent

l'excitació, la cavitat de ressonància i l'efecte de radiació dels llavis, permetent la seva anàlisi acústica i l'extracció dels paràmetres que possibiliten una aproximació al fenomen de la fonació amb suficient fidelitat.

La figura 2 mostra el model de producció de la veu. A continuació es descriu aquest model [7]:

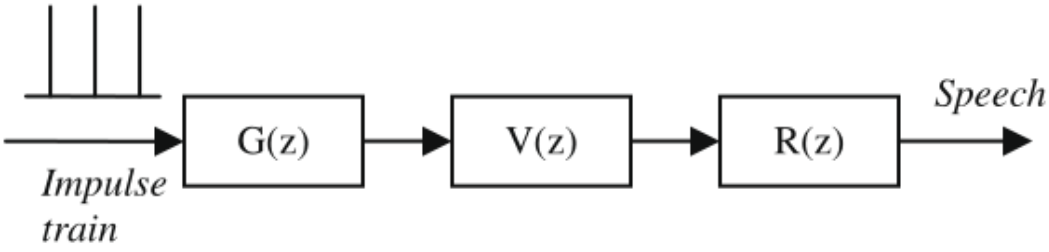

Figura 2: Model de producció de la veu [7]

S'utilitza el tren d'impulsos per modelar el ritme d'alenades d'aire a través de la glotis, les quals resulten de l'oscil·lació de les cordes vocals durant la fonació. Aquests polsos quasi periòdics són els que conformen la forma d'ona de la glotis, amb la qual s'obté la freqüència fonamental de l'orador. L'espectre de la parla *S(z)*, mesurat a la sortida dels llavis, pot expressar-se en el domini freqüencial complex com:

$$S(z)=G(z)V(z)R(z) \tag{1}$$

On, *G(z)* representa el model de font de la glotis, *V(z)* és la funció del sistema del tracte vocal i *R(z)* la funció de radiació als llavis. Per tant, si les funcions del tracte vocal i la radiació estan ben modelades, podem utilitzar un filtre invers per obtenir *G(z)*:

$$G(z)=\frac{S(z)}{V(z)R(z)} \tag{2}$$

2.2.2 Extracció de característiques del senyal de veu

Una de les dificultats més importants dels sistemes SER, i que alhora provoca més diferències d'opinions entre els investigadors, és l'extracció de característiques del senyal de veu. Determinar quines són òptimes per a la caracterització de les emocions és

un tema de debat obert. Alguns estudis [10] enumeren 3 qüestions clau pel que fa a l'extracció de característiques: regió d'anàlisi del senyal, tipus de característiques de la parla i el preprocessament i postprocessament.

2.2.2.1 Regió d'anàlisi del senyal

Hi ha diferència d'opinions entre quina és la regió d'anàlisi del senyal que presenta millors resultats per a l'extracció de característiques. Alguns experts prefereixen dividir el senyal en petites porcions (*frames*), de cada una de les quals s'extreu un vector de «característiques locals». Per altra banda, cada cop hi ha més investigadors que defensen que l'extracció «característiques globals» (és a dir, de tota la sentència sencera) permet caracteritzar millor les emocions.

Partint de la base que el senyal de veu no és estacionari, per tal de processar-lo és divideix en *frames* , els quals es poden considerar aproximadament estacionaris [11]. Les característiques locals fan referència a la informació prosòdica com el *pitch* o l'energia que es poden extreure d'aquests *frames*.

Per altra banda, les característiques globals es calculen com estadístiques de tot els senyal sencer. Aquestes aporten beneficis pel que fa a precisió i temps de computació en la classificació. A més, el nombre és menor i els algorismes de selecció de característiques i validació creuada és resolen amb molta més rapidesa. Per contra, les característiques globals són eficients únicament per reconèixer emocions amb excitació similar; es a dir, és pot distingir entre emocions amb un nivell d'excitació elevat (per exemple , enuig, por o alegria) i un nivell d'excitació baix (per exemple, tristesa) però, no per a distingir entre estats amb nivells d'excitació iguals com enuig i alegria. Un altre inconvenient és que amb aquest mètode d'anàlisi és perd tota la informació temporal del senyal.

El fet d'escollir entre característiques globals o locals determina el mètode de classificació que s'haurà d'emprar. En tots cas, hi ha exemples [12] de sistemes SER en els quals els vectors de característiques combinen ambdues opcions i utilitzen més d'un sistema de classificació.

2.2.2.2 Categories de característiques en la parla

Seguint el criteri aportat per Moataz El Ayadi, et al., [10], les característiques de la parla es poden dividir en 4 grups: continues, qualitatives, espectrals, TEO-based (*Teager Energy Operator*). Les característiques continues estan relacionades amb la freqüència fonamental (F_0), l'energia del senyal o informació temporal. Les qualitatives, estan lligades a la qualitat de la veu, la tensió o la aspresa. Les característiques espectrals juguen un paper molt important en les aplicacions de reconeixement de la parla i també de les emocions. A grans trets, s'extreuen filtrant el senyal amb filtres passabanda, usant com a criteri escales freqüencials no lineals (escala de Mel, per exemple) que representen la percepció real humana respecte a diferents tonalitats, i aplicant la transformada inversa de Fourier (IFT) a l'espectre del senyal resultant (cepstrum). Finalment, les característiques TEO-based es basen en teories de la mecànica de fluids, on es determina que la producció sonora es deguda a les interaccions de vòrtexs no estacionaris produïts al tracte vocal, i són molt útils per la detecció d'emocions que comporten una veu amb alt grau de tensió. La figura 3 mostra un resum d'aquestes categories i les característiques que se'n poden extreure.

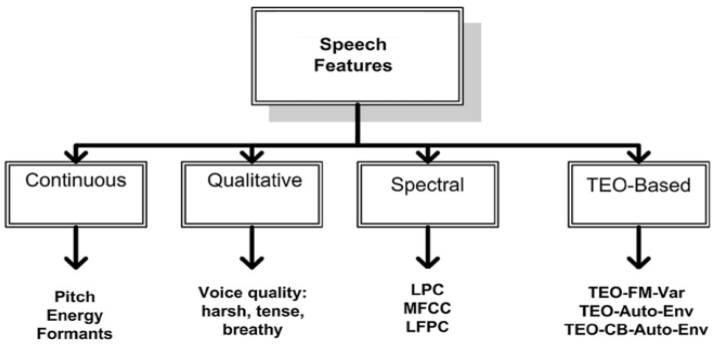

Figura 3:Categories de característiques de la veu [10]

De totes aquestes, la majoria d'estudis coincideixen en que gran part de la informació emocional de la parla es pot deduir de les característiques continues com el *pitch* i l'energia, així com de les variacions temporals que aquestes sofreixen. Altres estudis [13], també remarquen que les característiques espectrals que millors resultats donen per

al reconeixement d'emocions en la parla són les LFPC, envers a les MFCC (molt utilitzades en la parametrització de l'àudio en general), degut a que conserven millor la informació relacionada en la freqüència fonamental en els filtres d'ordre inferior.

A nivell pràctic [14], es calcula el *pitch* i l'energia de cada *frame* de cada oració emocionada. Aquest resultats s'emmagatzemen en forma de vectors; cada un etiquetat amb l'emoció corresponent. Una pràctica habitual és el calcul de la primera derivada d'aquests vectors per tal d'obtenir la variació soferta a nivell temporal. De cada vector obtingut se'n poden calcular certs paràmetres com: valor màxim, mínim, marge, desviació estàndard, mitjana, quartils, *interquartile range*, *skewness* i *kurtosis*.

Pel que fa a les característiques LFPC (short time *Log Frequency Power Coefficients*) [13], s'extreuen amb un banc de filtres que divideixen cada *frame* en 12 bandes freqüencials (veure la figura 4), les quals coincideixen amb les bandes de percepció de l'orella humana. Com s'explica en el següent apartat, abans d'aplicar el banc de filtres s'aplica l'enfinestrat i la FFT[1]. S'obté, finalment, un vector amb 12 coeficients LFPC per a cada *frame*.

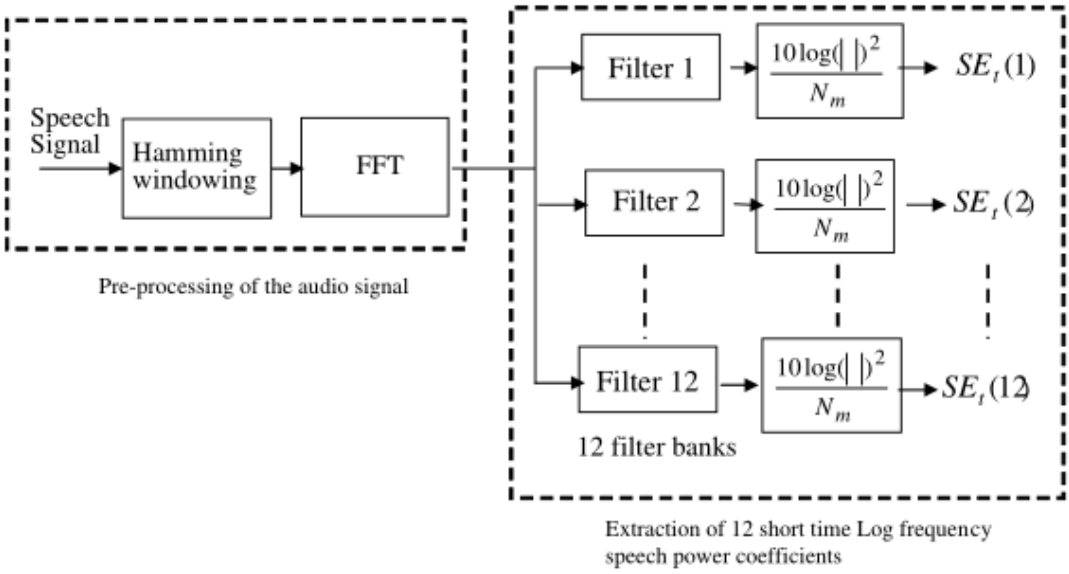

Figura 4: Esquema d'extracció de característiques LFPC [13]

1 La FFT (*Fast Fourier Transform*) o transformada ràpida de Fourier és un mètode computacional per calcular de manera molt ràpida la transformada discreta de Fourier (DFT, *Discret Fourier Transform*). En termes d'eficiència, la DFT té un cost temporal de $O(n^2)$ i la FFT de O(nlog n). Per poder aplicar la transformada ràpida de Fourier, es necessiten dues condicions indispensables: que el senyal discret a transformar sigui periòdic i que el nombre de mostres sigui de l'ordre de 2^n, amb *n* essent un nombre enter positiu

2.2.2.3 Preprocessament i postprocessament

El preprocessament fa referència a totes aquelles operacions realitzades sobre el senyal enregistrat, necessàries per afrontar l'extracció de característiques. La metodologia habitual [11] en el preprocessament dels senyal de veu segueix els passos següents:

1. Filtratge de freqüències no desitjades afegides per l'entorn o l'equipament emprat alhora de fer al captació, així com la normalització energètica per evitar diferències entre les diferents captacions. En aquest punt també s'inclou l'eliminació de parts «no necessàries», com els silencis. Una pràctica molta habitual als sistemes de reconeixement de la parla (ASR), tot i que no ho és en el cas dels sistemes SER [13].

2. Partint del model font-filtre descrit anteriorment, obtenció de les funcions de transferència de radiació *R(z)* i de tracte vocal *V(z)* , per tal d'aconseguir diferents components del model de veu.

3. Segmentació en blocs i enfinestrat. Per a l'extracció de característiques locals es divideix el senyal en petits blocs que s'anomenen finestres, la longitud dels quals varia entre 20 i 130 ms. Els blocs s'han d'encavalcar entre ells per aconseguir millor correlació entre les característiques dels segments adjacents. Per tal d'evitar distorsions degudes a la no periodicitat dels blocs, aquests segments es multipliquen per finestres Hamming, les quals aproximen els valors inicial i final a zero i accentuen els valors centrals. A cada un d'aquests blocs se li ha d'aplicar la FFT per fer la caracterització a nivell espectral. La figura 5 mostra el resultat d'aplicar l'enfinestrat i la posterior FFT a un senyal [9].

Per altra banda, el postprocessament és el pas previ al entrenament i testeig del sistema de classificació. El primer pas és l'estandardització dels valors de les característiques dins del mateix ordre de magnitud. També és útil per reduir l'emmagatzematge de dades i, en conseqüència, disminuir els requeriments computacionals dels sistema. S'utilitza la selecció de característiques basada en filtres per reduir el nombre d'aquestes. Això permet obtenir vectors amb el nombre mínim de característiques que ofereixi el màxim d'informació possible.

La selecció de característiques es pot usar per identificar els atributs innecessaris, irrellevants i redundants del conjunt de dades. En aplicar mesures estadístiques, pot determinar quines columnes no contribueixen a la precisió del model de predicció (o, de fet, podrien disminuir la precisió del model) i treure-les abans d'entrenar un model.

La selecció de característiques basada en filtres usa diferents proves estadístiques per determinar el subconjunt de característiques que tenen la màxima eficàcia predictiva. Es tria una mesura estadística per aplicar i el mòdul calcula una puntuació per a cada columna que s'ha fet servir com una característica. A continuació, les característiques es classifiquen per la puntuació i les columnes de característica, de manera que les millors puntuacions s'usen en la creació del model i la resta es mantenen en el conjunt de dades però no es fan servir per a l'anàlisi.

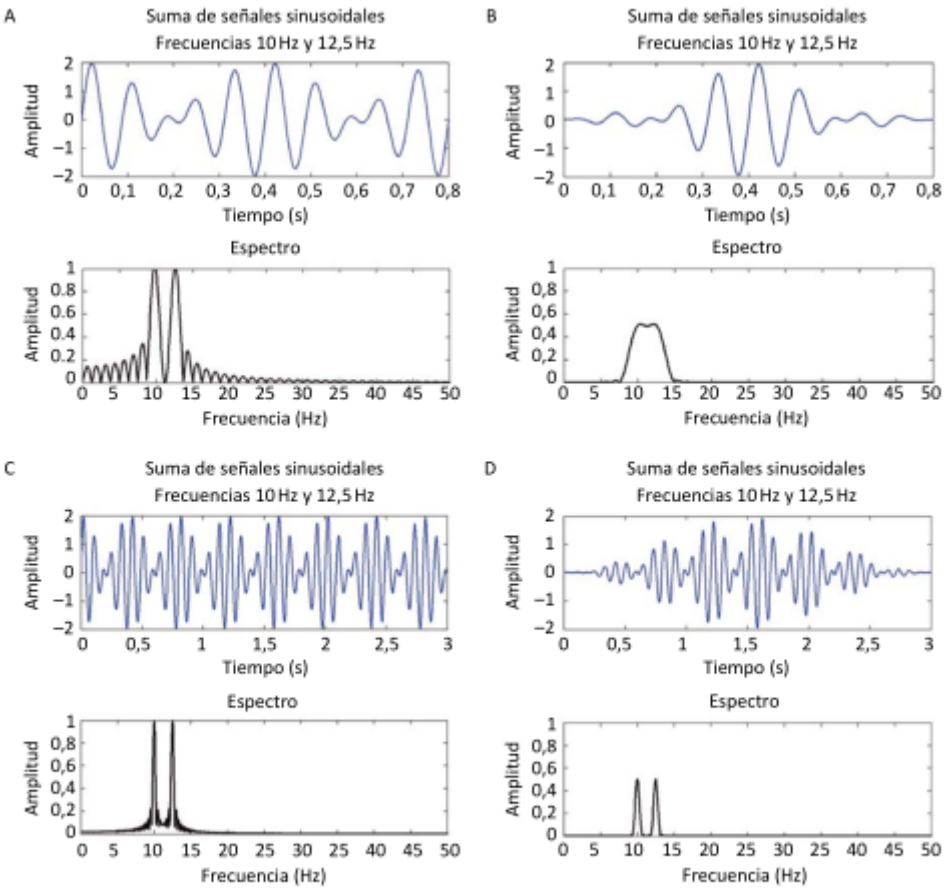

Figura 9. Efecto del troceado y enventanado de la señal sobre una señal compuesta por la suma de dos sinusoides. A) Espectro con ventana rectangular de duración 0,8 s. B) Espectro con ventana de *Hamming* de duración 0,8 s. C) Espectro con ventana rectangular de duración 3 s. D) Espectro con ventana de *Hamming* de duración 3 s.

Figura 5: Segmentació i enfinestrat del senyal [9]

2.3 Sistemes de classificació

Un sistema de classificació és un tipus de mecanisme d'aprenentatge supervisat, la finalitat del qual és decidir quina sortida al sistema és la més adequada segons un model de decisió predeterminat. Aquest sistema utilitza algorismes que «aprenen» a classificar noves observacions donat exemples de dades prèviament etiquetades. Hi ha una fase d'entrenament del sistema, on s'etiqueten les dades i es crea un model que servirà per a supervisar les noves observacions.

El disseny dels sistemes SER consta de dos blocs bàsics [10]: (1) un sistema *front-end* que extreu les característiques de la parla de les dades disponibles (la qual s'ha estudiat fins ara) i, (2) un sistema de classificació que decideix quina és l'emoció subjacent en una trama de veu donada. Aquesta segona etapa és tant o més important que la primera. De fet, molts dels estudis en l'àmbit del reconeixement d'emocions en la parla se centren en la recerca de nous mètodes de classificació, o la combinació de varis d'aquests [7] [15][16].

Hi a diversos sistemes de classificació emprats en els sistemes SER: HMM, GMM, k-NN, ANN, SVM, entre d'altres. Hi ha força contradiccions entre quin és el millor sistema de classificació per al reconeixement d'emocions. Per exemple, es poden trobar treballs [7] amb molt bons resultats on s'usen les característiques MFCC amb sistemes de classificació propis, en aquest cas OPF (*Optimum-Path Forest*); la figura 6 mostra el resultat comparatiu amb d'altres mètodes. Per contra, altres estudis [13], donen molts bons resultats amb característiques LFPC i sistema de classificació HMM. Es pot concloure, en conseqüència, que escollir un sistema de classificació o un altre dependrà de les característiques extretes i de la finalitat del projecte.

En aquesta fase també es valida el model creat mitjançant la validació creuada de k iteracions (*k-fold cross-validation*). El procediment és el següent: les dades de mostra es divideixen en k subconjunts. Un dels subconjunts s'utilitza com a dades de prova i la resta (k-1) com a dades d'entrenament. El procés de validació creuada es va repetint durant les k iteracions, amb cada un dels possibles subconjunts de dades de prova. Finalment es realitza la mitjana aritmètica dels resultats de cada iteració per obtenir un únic resultat. Aquest mètode és molt precís ja que s'avaluen a partir de k combinacions

de dades d'entrenament i de prova però, tot i així, té un desavantatge, i és que, a diferència del mètode de retenció, és lent des del punt de vista computacional. A la pràctica, l'elecció del nombre d'iteracions depèn de la mesura del conjunt de dades.

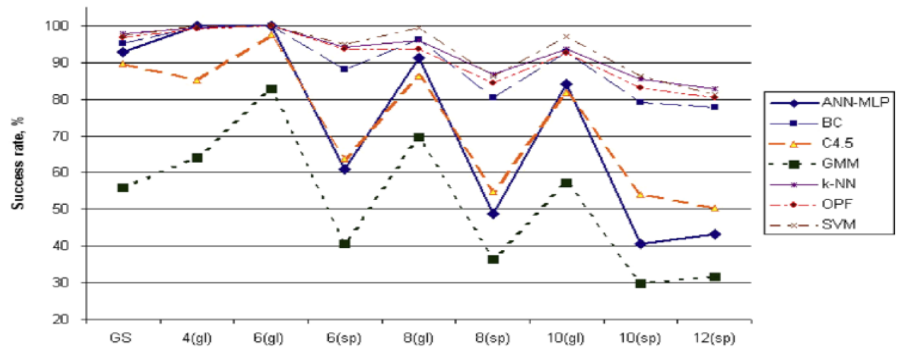

Fig. 5. Mean recognition rates for seven classification systems for ANGRY versus HAPPY versus SAD versus NEUTRAL after 10 rounds for glottal symmetry (GS), glottal MFCC (#(gl) and speech MFCC (#(sp)).

Figura 6: Comparativa entre mètodes de classificació [7]

A continuació es fa una breu explicació d'aquells sistemes de classificació que aporten millors resultats per al reconeixement d'emocions en la veu, segons les referències d'aquest document. És descriuen a nivell teòric i de forma resumida els sistemes SVM, HMM i k-NN.

2.3.1 Support Vector Machines (SVM)

Aquest sistema permet separar un conjunt de dades mitjançant separadors lineals o hiperplans, de tal manera que els diferents grups quedin separats per una distància màxima, que s'anomena marge màxim. Utilitzant l'hiperplà obtingut es poden classificar les dades d'entrada noves en el grup corresponent.

A la figura 7 es veu un exemple senzill de 2 dimensions, en què el hiperplà per realitzar la classificació és d'una dimensió (una línia). Hi ha diverses línies o hiperplans que realitzen la classificació, però l'òptim és aquell en el qual el marge màxim és major. Els vectors de suport en què es basa són les dues línies o plans paral·lels a la línia o hiperplà òptim, a una distància el valor de la qual és el marge òptim; tal com s'observa en la segona imatge de la figura 7.

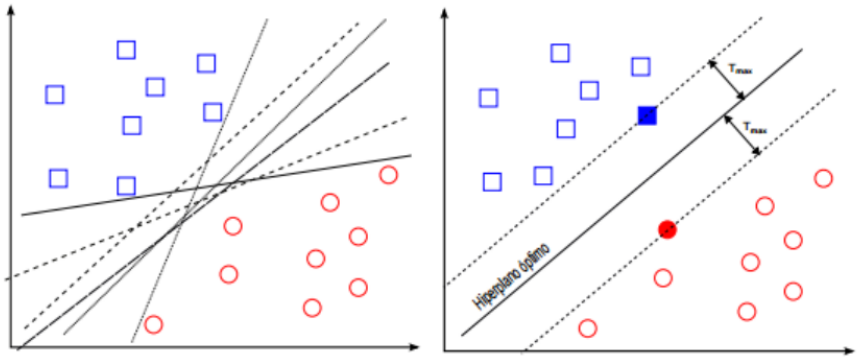

Figura 7: Exemple de classificació amb SVM

En problemes reals, mai o gairebé mai es donarà un cas on la separació a realitzar es faci mitjançant una línia recta o hiperplà. Per tant, s'ha de fer un mapatge de la informació del pla actual a un espai de característiques de major dimensió. Aquest mapatge es realitza amb l'ajuda d'unes funcions anomenades funcions de nucli o *kernell*.

2.3.2 Hidden Markov Models (HMM)

És un model estadístic que permet modelar un procés de Markov on els paràmetres són desconeguts amb l'ajuda dels paràmetres observables. En models de Markov normals les probabilitats de transició entre estats són els únics paràmetres del model, ja que els estats són visibles. Però, en el cas dels models ocults els estats no són visibles. Amb la qual cosa, aquest mètode permet obtenir els estats ocults més probables a partir dels observables.

La figura 8 mostra un HMM simple on *x(t)* és el valor d'una variable aleatòria en l'instant de temps *t*, i *y(t)* és la variable observada en el mateix instant de temps *t*. Es pot deduir que el valor actual de la variable aleatòria oculta només depèn del seu valor anterior i, d'igual manera, el valor de la variable observada només depèn de la variable oculta.

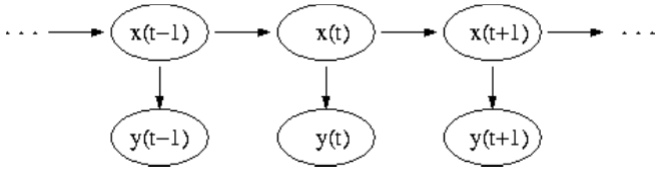

Figura 8: Exemple de classificació amb HMM

Un sistema HMM (veure la figura 9) està definit pels següents paràmetres:

- Conjunt d'estats ocults (x).
- Conjunt de possibles valors observables de cada estat (y).
- Distribució de probabilitat de l'estat inicial.
- Distribució de probabilitat de la transició entre estats (a) .
- Distribució de probabilitat de les observacions en cada estat (b).

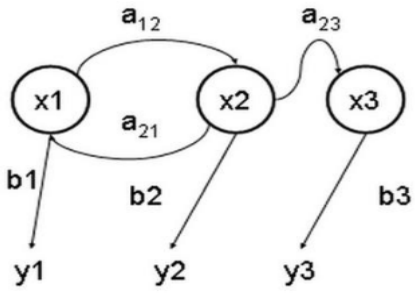

Figura 9: Exemple d'estats d'un sistema HMM

2.3.3 k-Nearest Neightbors (k-NN)

El mètode k-NN mesura la distància entre un ítem desconegut i un conjunt d'ítems coneguts que formen el model. Tant l'ítem desconegut com els ítems del model estan formats pel mateix tipus de característiques. Es basa en escollir els *k* ítems del model en què la distància amb l'ítem desconegut sigui mínima. Com l'ítem del model està identificat (etiquetat), aquesta etiqueta correspondrà a la identificació per l'ítem desconegut. Existeixen múltiples formes de calcular la mesura de la distància. La més habitual i la que es la distància Euclidiana. La seva expressió és:

$$d(x,y)=\sqrt{\sum_{j=1}^{N}(x_j-y_j)^2} \tag{3}$$

On x_j i y_j són la característica j dels ítems *x* i *y*. N és el nombre total de característiques.

2.4 Obtenció de dades

Per tal d'obtenir els millors resultats possibles i evitar riscos durant el desenvolupament del projecte, s'ha de comença amb una base sòlida. Aquesta base són les dades. Per a aquest treball, les dades han de ser oracions de diferents persones i diferents gèneres, i cada una d'aquestes amb diferents emocions. A més a més, l'obtenció d'aquestes dades (captació i emmagatzematge) ha de seguir certs criteris de qualitat, que s'enumeren a continuació:

- **L'entorn de captació ha de ser adequat**. La sala ha de complir amb uns mínims d'acondiciament acústic, per a evitar reflexions que podrien malmetre el posterior anàlisi. També es necessari l'ús de material de gravació de certa qualitat per evitar distorsions afegides; microfonia, targeta de conversió A/D i ordinador i programari d'edició i captació.
- **Mostreig del senyal i quantificació**. És necessària una freqüència de mostreig de com a mínim el doble de l'amplada de banda del senyal a mostrejar (criteri de Nyquist). Suposant que l'amplitud de l'espectre per a la veu humana decreix considerablement a partir dels 8-10 kHz, la freqüència de mostreig mínima ha de ser de 20 kHz [9]. Per altra banda, i a grans trets, la quantificació permet representar l'amplitud del senyal a diferents nivells (marge dinàmic). Suposant que el marge dinàmic de la veu és de 85 dB [9],la quantificació ha de ser de 16 bits; amb la qual es permet un marge dinàmic de 90 dB.
- **Fidelitat de les emocions.** La millor opció és tenir veus amb emocions reals, no actuades. Però, aconseguir-ho és realment costos. En conseqüència, en la majoria d'estudis es fa referència a bases de dades amb emocions actuades.

Per a la creació de la base de dades hi ha dues opcions: produir la base de dades des de cero o aconseguir-ne una. Tenint en compte els requeriments esmentats anteriorment, la primera opció no és viable a curt termini. Per tant, s'ha optat per la segona; tot i que no ha estat fàcil ja que la majoria de bases de dades per a finalitats del reconeixement d'emocions no són gratuïtes o, fins i tot, no presenten la qualitat desitjada. Una possibilitat és la base de dades RAVDESS creada per SMARTLAB i de descàrrega i ús gratuït sota la llicència *Creative Commons* [17] .

RAVDESS conté fitxes d'àudio i vídeo amb veus parlades i cantades de 12 actors (6 homes i 6 dones) amb diferents emocions, amb idioma angles i accent nord-americà. En concret, per a aquest treball s'ha seleccionat les veus parlades. La recopilació conté un total de 4.320 fitxers, amb un mostreig de 48 kHz i 16 bits de quantificació. A continuació es llisten les especificacions:

- Nombre d'actors: 12
- Gènere (2: masculí, femení)
- Emoció (7: calma, alegria, tristesa, enuig, por, disgust i sorpresa) + neutre
- Intensitat emocional (2: normal, fort)
- Oracions (2, «*Kids are talking by the door*», «*Dogs are sitting by the door*»)
- Nombre de repeticions: 2

Les emocions han estat gravades amb dos nivells d'intensitat, exceptuant l'estat neutre.

Capítol 3 Disseny

Partint de l'estudi anterior, s'ha planificat un disseny modular que permeti la implementació per blocs, de manera que se simplifiqui el funcionament i atorgui flexibilitat i escalabilitat al sistema.

En el següent diagrama es presenta l'estructura bàsica del projecte.

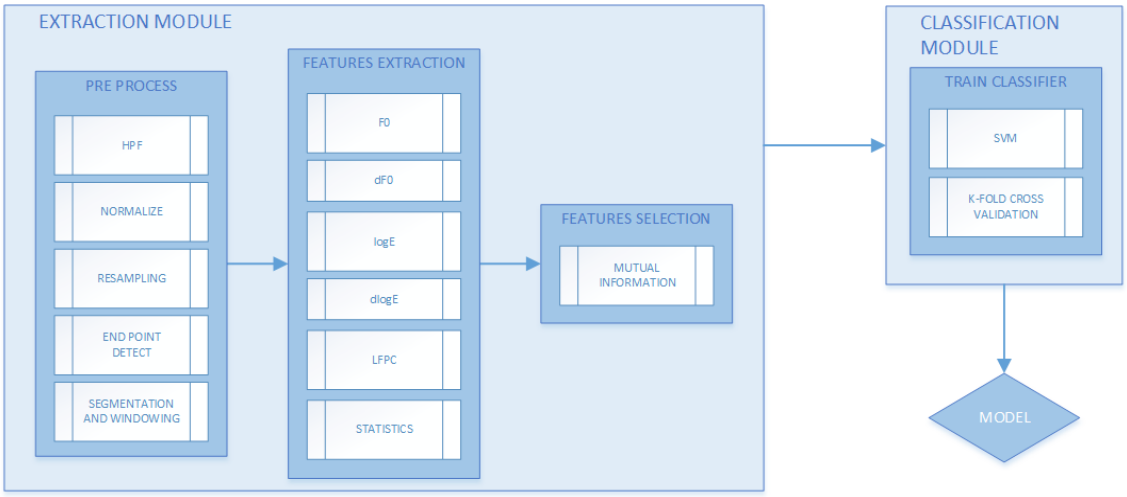

Figura 10: Diagrama de blocs del projecte

L'arquitectura presentada consta bàsicament de dos blocs principals: el modul d'extracció de característiques (*extracction module)* i el mòdul d'entrenament del sistema (*classification module),* del qual s'obté el model de classificació. Cada un d'aquests mòduls es compon de submòduls amb els processos corresponents; tal i com mostra la figura 10. Aquests processos s'implementen en forma de funcions amb el programari MATLAB, descrites en els capítols d'implementació.

El mòdul d'extracció de característiques s'encarrega del processament del senyal que entra al sistema. A la sortida d'aquest mòdul (entrada al mòdul de classificació) s'obtenen tots els paràmetres estadístics de cada frase i una llista de característiques ordenades de més a menys rellevància. Per dur a terme aquesta tasca, el disseny incorpora tres submòduls: preprocessament, extracció de característiques i selecció de característiques. En el primer es llegeixen la dades d'entrada i es preparen per tal de poder-ne extreure les característiques desitjades. A la sortida s'obté una matriu que és l'entrada al bloc d'extracció de característiques, del qual s'obté un vector per cada característica: *pitch*, derivada del *pitch*, energia, derivada de l'energia i 12 coeficients LFPC. Un total de 16 vectors que representen els valors de cada segment per a cada característica. Finalment, de cada vector s'obtenen 12 paràmetres estadístics: *mean, min, max, standard deviation*, *skewness, kurtosis*, *1st quartile, 2nd quartile, 3rd quartile, interquartile range, semi-interquartile deviation* i *90 percentile.* La sortida del submòdul d'extracció de característiques és una taula amb 16x12=192 paràmetres (columnes) per a cada frase (files). Aquesta taula és l'entrada al bloc de selecció de característiques, del qual, mitjançant la selecció basada en filtres (en concret amb el mètode d'informació mútua), s'obté la llista de paràmetres ordenats de més a menys rellevància. Cal afegir que en l'estudi previ s'ha incorporat el procès de selecció de característiques i estandarditzación dels valors d'aquestes dins del postprocessament; no s'ha dissenyat en aquest projecte un procés d'estandarditzación ja que aquest s'inclou en el sistema de classificació escollit (SVM).

El mòdul de classificació rep la taula i la llista de paràmetres ordenats per rellevància. Amb aquestes dades el sistema de classificació s'entrena i realitza la validació amb el mètode *k-fold cross validation*. La sortida d'aquest bloc és un model que servirà per a fer la predicció de les dades desconegudes.

Com es veurà en el següent punt, hi ha dos fases d'execució del sistema: entrenament i predicció. En la primera s'introdueixen les dades conegudes al sistema i es fa ús dels dos mòduls, obtenint el model de classificació esmentat. En la segona, es fa la predicció de les dades desconegudes utilitzant el model i el mòdul d'extracció però no el mòdul de classificació.

La idea principal d'aquest disseny és permetre flexibilitat alhora de realitzar la implementació. Es pot crear una aplicació bàsica amb els paràmetres prefixats o bé, una de més complexa, que permeti parametritzar cada bloc de manera que es pugui modificar el comportament del sistema i adaptar-lo a les necessitats. A més, el disseny modular dona un cert grau d'escalabilitat degut a l'abstracció entre entre els processos i mòduls; es poden afegir o eliminar processos de manera relativament senzilla sense alterar el funcionament bàsic del sistema.

Capítol 4 Fases d'execució del sistema

Tot i que les fases d'execució són l'últim pas en la implementació, es precís explicar-ho amb antelació per tal de comprendre millor el funcionament general del sistema. A continuació es presenten aquestes fases.

4.1 Fase d'entrenament

En aquesta fase el sistema recull les dades desconegudes i les introdueix al sistema, les processa i entrena el classificador per a obtenir el model. El següent diagrama mostra el funcionament:

Figura 11: Diagrama de la fase d'entrenament del sistema

Aquesta fase s'executa des de MATLAB amb la funció **SER_TRAINING.m**. L'entrada a aquesta funció és el directori principal on hi ha les dades d'entrenament. A més, també permet l'entrada opcional d'alguns paràmetres:

- freqüència de mostreig (**fs**); valor per defecte: 22050 Hz
- temps de finestra (**frame_size**); valor per defecte: 0,023 segons
- freqüència de tall del filtre (**HPF_fc**);valor per defecte: 100 Hz
- identificador de fitxer (**fid**); veure capítol 7
- referència a objecte de monitoratge de la GUI (**CTSout**); veure capítol 7

Aquests paràmetres afecten directament al submòdul de preprocessament i, depenent dels valors que prenguin, el resultat pot variar de manera significativa, amb la qual cosa es poden realitzar multitud de proves. Els valors recomanats i l'efecte d'aquests al sistema s'explica en l'apartat d'implementació del mòdul d'extracció. Una millora del sistema seria la possibilitat d'introduir més paràmetres per variar el comportament d'altres submòduls; per exemple, l'opció d'escollir entre les característiques a extreure i els paràmetres estadístics d'aquestes. Això permetria incrementar el nombre d'experiments i optimitzar el sistema.

Les variables d'entrada **fid** i **CTSout** s'utilitzen exclusivament en l'execució de la interfície gràfica (veure capítol 7) i, per tant, no té sentit parametritzar-les de manera manual des de la línia de comandes de MATLAB. En conseqüència, s'exclouen dels capítols d'implementació 5 i 6.

Durant l'execució, es mostra per la finestra de comandament de MATLAB el progrés i el resultat amb el percentatge de precisió del model i el nombre de característiques utilitzades. Finalment, es genera la matriu de confusió amb la funció **heatmap.m**; aquesta no s'inclou en la versió R2016a, per aquest motiu s'adjunta amb la resta de funcions del projecte.

A la sortida de la funció SER_TRAINING.m s'obté el model amb millor percentatge. Aquest model conté tota la informació necessària per a realitzar la predicció. Els paràmetres opcionals es guarden en el camp *opts;* per tal de processar l'àudio d'entrada de la fase de predicció de manera idèntica als àudios utilitzats per entrenar el sistema. També, s'afegeix el camp *Accu* on es guarda el valor de precisió del model.

4.1.1 Càrrega de dades al sistema

Per tal de carregar les dades conegudes al sistema, en aquesta fase s'executa la funció auxiliar **LOADDATA.m**, que rep com a paràmetre d'entrada el directori principal i obté a la sortida una estructura amb les classes i els noms dels fitxers d'àudio per a cada una d'elles. Per a garantir un funcionament correcte, aquest directori ha de seguir una estructura concreta:

1. Dins del directori principal, els àudios s'han de separar per carpetes independents segons l'emoció.

2. Cada carpeta ha de contindre el mateix nombre de fitxers. El nom de la carpeta determina l'etiqueta de cada classe.

Això vol dir que el sistema és independent de la base de dades amb que es treballi. En aquest projecte els àudios s'obtenen de la base de dades RAVDESS. Aquesta codifica el nom de cada fitxer de manera que és senzill utilitzar diferents emocions, separar per generes o intensitats. El format de la codificació es troba en el fitxer *readme_ravdess.txt.* Amb això es poden provar experiments amb conjunts de diferents emocions o estudiar com afecta la intensitat o el genere en la identificació d'emocions o, inclús, provar el sistema amb altres bases de dades de manera senzilla.

4.2 Fase de predicció

En aquesta fase el sistema prediu, segons el model donat, quina és l'emoció d'un fitxer d'àudio determinat. En aquest cas no s'utilitza el mòdul de classificació. El següent diagrama mostra el funcionament:

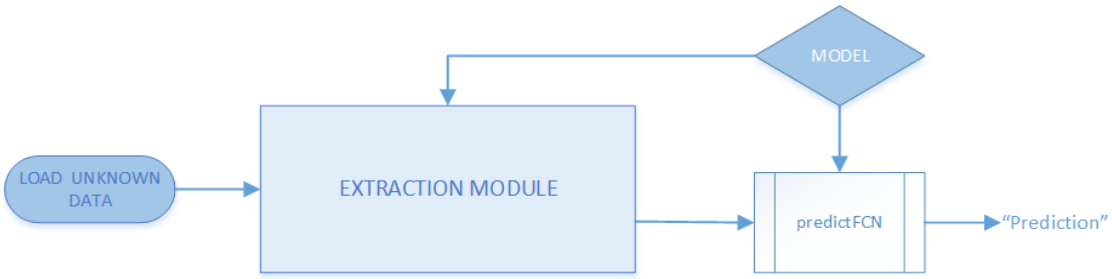

Figura 12: Diagrama de la fase de predicció del sistema

Aquesta fase s'executa amb la funció **SER_PREDICT.m**. A l'entrada d'aquesta funció és necessari especificar el nom del fitxer a analitzar i el model amb el qual s'ha de classificar. Aquest model, com s'ha dit, conté els paràmetres opcionals per tal de processar l'àudio de manera correcta.

Per poder utilitzar el modul d'extracció amb un nom de fitxer (enlloc d'una estructura com en la fase anterior), aquest àudio es carrega al sistema amb la funció **LOADFILE.m**.

La predicció a la sortida s'obté amb la funció *predictFcn* que es crea de manera automàtica amb el model.

Capítol 5 Implementació: Mòdul d'extracció de característiques

Aquest mòdul permet la parametrització dels fitxers d'àudio que entren al sistema. S'executa durant les dues fases del punt anterior (entrenament i predicció) mitjançant la funció **EXTRACTION_MODULE.m**. Aquesta, rep l'estructura procedent de la funció LOADDATA.m (en el cas de la fase d'entrenament) o de LOADFILE.m (en el cas de la fase de predicció); la resta de paràmetres són els opcionals (**fs**, **HPF_fc**, **frame_size**) que s'introdueixen manualment en la funció SER_TRAINING.m. Les sortides són:

1. Una taula amb tantes files com fitxers s'hagin introduït (observacions) i 194 columnes; de les quals les 2 primeres són les etiquetes (numero i nom) i la resta són els paràmetres estadístics

2. Un vector amb les característiques (o paràmetres) ordenades per rellevància segons l'algorisme emprat en el submòdul de selecció de característiques.

5.1 Preprocessament

El submòdul de preprocessament s'encarrega de preparar cada frase per a l'anàlisi de característiques. La funció **PREPROCESS.m** rep un nom de fitxer, els paràmetres opcionals i el temps d'encavalcament entre finestres; que es calcula al mòdul d'extracció. A la sortida d'aquest bloc s'obté una matriu per cada àudio d'entrada on les files representen cada segment (*frame)* i les columnes el nombre de mostres; el nombre de files i columnes dependrà de la freqüència de mostreig i de la duració de la finestra (com s'explica en el punt 5.1.5 Segmentació i enfinestrament). Cada casella, per tant, representa el valor de la mostra després d'aplicar l'enfinestrament amb la funció *Hamming.*

5.1.1 Filtratge

Durant el procés d'enregistrament es poden generar sorolls que provocarien un mal comportament del sistema. Aquests sorolls solen ser deguts a senyals de continua dels instruments emprats en les gravacions (micròfons, cablejat, interfícies d'àudio, convertidors A/D,...) i acostumen a estar per sota dels 100 Hz. S'ha dissenyat un filtre amb la finalitat d'eliminar aquesta banda freqüencial (es pot introduir una altra freqüència de tall de manera opcional) que s'executa dins la funció **hpf.m**. Aquesta funció també calcula la freqüència de tall normalitzada (FCN) amb els paràmetres que rep a l'entrada: freqüència de tall i de mostreig.

El filtre que millor resposta ofereix en la banda de transició és el filtre el·líptic. Per calcular els zeros i pols es dissenya un filtre d'ordre 6, amb 0.1dB d'arrissament en la banda de pas i 60dB de caiguda en la transició. Amb MATLAB es pot calcular amb la funció **ellip**:

$$[z, p] = ellip(6, 0.1, 60, FCN, 'high') \tag{4}$$

FCN es calcula amb:

$$FCN = \frac{\frac{2\pi fc}{fs}}{\pi} \tag{5}$$

on *fc* és la freqüència de tall del filtre i *fs* la freqüència de mostreig. La resposta d'aquest filtre és:

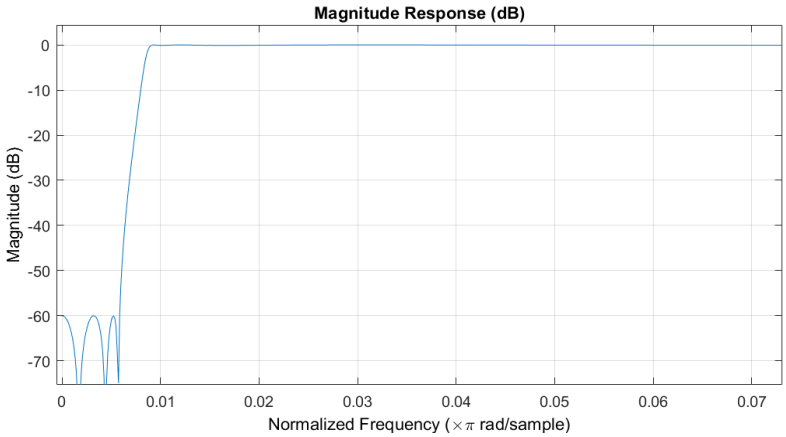

Figura 13: Resposta del filtre passa alt

5.1.2 Normalització

Aquest procés normalitza l'energia del senyal que entra al sistema en el rang de valors [-1,1]. Amb això s'aconsegueix evitar diferències entre captacions i solucionar problemes en les prediccions degudes a diferències energètiques del senyal.

El senyal normalitzat es calcula dividint cada valor de cada mostra pel valor màxim de totes les mostres, prenent els valors absoluts. Amb MATLAB s'utilitza la següent expressió:

$$y = x / max(\text{abs}(x)) \tag{6}$$

5.1.3 Remostreig

El sistema ofereix la possibilitat de canviar la freqüència de mostreig en cas que la de l'àudio original sigui més gran que l'especificada com a paràmetre. És a dir, es realitza una disminució de les mostres però mai el cas contrari; fet que no tindria sentit en aquest projecte.

Per defecte es mostreja a 22050 Hz, freqüència amb la qual es respecta el teorema de Nyquist[2] respecte a l'ample de banda freqüencial d'un senyal de veu, que conté informació valuosa fins a aproximadament 10000 Hz.

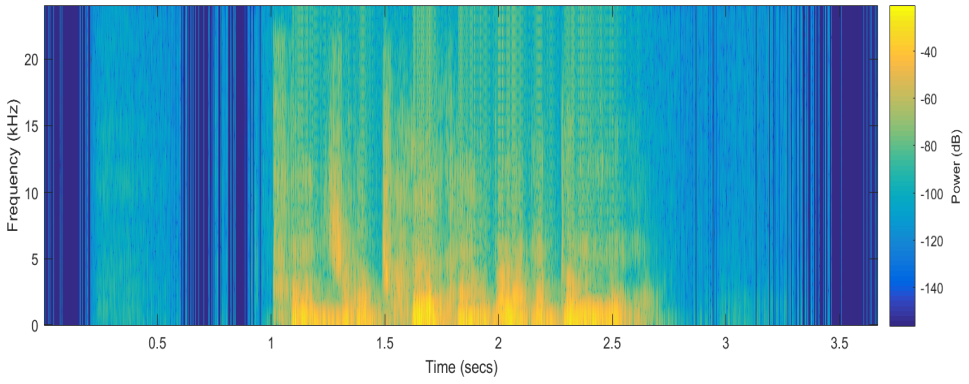

Figura 14: Espectrograma d'un àudio de la base de dades RAVDESS

2 Aquest teorema diu que la freqüència de mostreig ha de ser com a mínim el doble de l'amplada de banda del senyal a mostrejar. Per tant, si es desitja processar informació fins als 10000 Hz es necessària una freqüència de mostreig de com a mínim 20000 Hz. La freqüència de mostreig estàndard per a àudio més pròxima a aquest valor és la de 22050 Hz.

La figura 14 mostra l'espectre d'un àudio de la base de dades RAVDESS (48000 Hz de mostreig). Es pot apreciar com per sobre dels 10000 Hz el senyal perd amplitud progressivament i per tant aporta menys informació.

La llibreria *Signal Processing Toolbox* de MATLAB conté una funció específica amb aquest propòsit:

$$y = resample(x, p, q) \tag{7}$$

on y és el senyal x remostrejat en un factor p/q vegades. El valors de p i q els proporciona la funció **rat**:

$$[p, q] = rat(fs/fs_o) \tag{8}$$

on fs és la freqüència de mostreig donada i fs_o és la freqüència del senyal.

5.1.4 End Point Detect

Un pas determinant per a la correcta extracció de característiques és la detecció de l'inici i final de la veu dins del fragment d'àudio. Amb això s'aconsegueix eliminar els silencis del principi i final de les gravacions, que condicionarien de forma important els valors de les característiques que s'extreuen en el següent submòdul.

Per fer-ho s'ha utilitzat la funció **endPointDetect.m** (Roger Jang, 2004 -amb modificacions per a aquest projecte-). Aquest funció retorna les posicions d'inici i final dels segments de veu d'un àudio determinat, així com la posició d'inici i final de la frase. En aquest projecte és important no eliminar tots els segments de veu que no fan referència a la fonètica, ja que per a la identificació d'emocions és necessari tenir en compte els elements prosòdics de la parla. Amb la qual cosa interessen, únicament, les posicions d'inici i final de la frase.

La detecció d'aquestes posicions es determina segmentant l'àudio en *frames* d'un numero determinat de mostres. De cada *frame* es calcula el volum i la taxa de pas per zero (ZCR , *Zero Crossing Rate*). La ZCR determina la quantitat de vegades que un senyal (en aquest cas *frame*) passa d'un valor negatiu a un positiu i viceversa. En els fragments de veu, la ZCR pren valors petits i en els silencis pren valors elevats. Per

contra, el volum és més elevat en fragments parlats i menys en els silencis. Creuant els resultats i comparant-los amb els dels *frames* propers, es pot fer l'estimació dels punts de transició entre veu i silenci. La següent imatge mostra el funcionament d'aquesta funció:

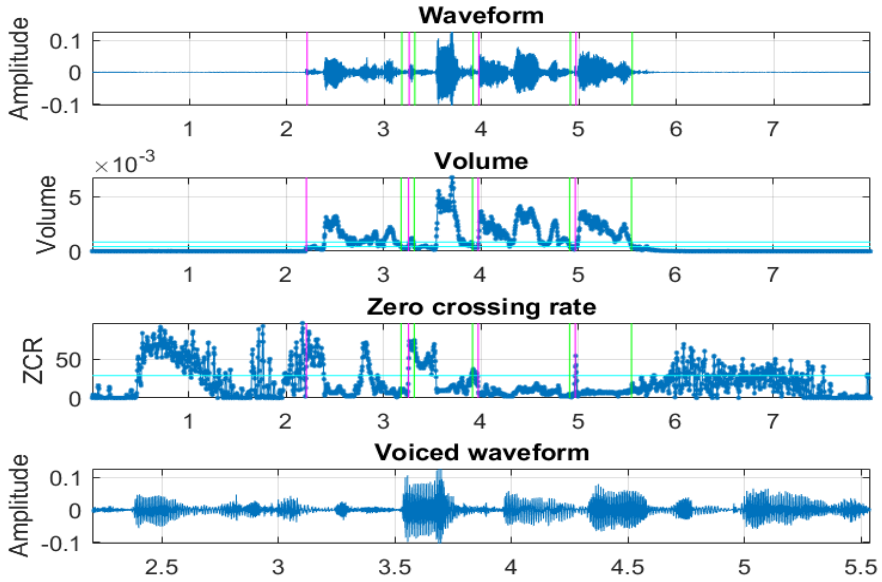

Figura 15: Funcionament endPointDetect

La imatge mostra el resultat sobre un àudio aleatori de la base de dades RAVDESS. Es pot apreciar com el volum és gairebé zero en les zones de silenci, mentre que la ZCR pren valors elevats. En els cas de les zones amb veu passa el contrari. L'última forma d'ona correspon a l'àudio sense aquests fragments de silenci.

5.1.5 Segmentació i enfinestrament

Per a l'anàlisi del *pitch,* l'energia i els coeficients LFPC (veure secció 5.2) és necessari processar senyals de caràcter estacionari. Donada la naturalesa no estacionaria del senyal de la parla, es precís separar-lo en petits segments que es poden considerar estacionaris.

La funció que realitza aquest procés és **windowing.m**. Aquesta funció rep els paràmetres opcionals de temps de finestra, temps d'encavalcament i freqüència de mostreig, que determinen el nombre de segments i les mostres de cada un d'ells. El temps d'encavalcament es calcula de manera automàtica (en el mòdul anterior) dividint el temps de finestra entre 2, proporcionant la correlació necessària entre els blocs.

Tot i que aquests paràmetres són opcionals, és important saber que la FFT, que es calcula per a l'anàlisi espectral de cada segment del senyal, treballa sobre un nombre de mostres igual a una potència de 2. Si el nombre de mostres del segment no arriba a una potència de 2, la FFT afegeix zeros (*zero padding*) fins al següent nombre «potència de 2». Per tant, és important escollir un temps de finestra que, en funció de la freqüència de mostreig, determini un nombre de mostres el més aproximat a una potència de 2 sense sobrepassar-la.

La taula 2 mostra els temps de finestra recomanats:

Taula 2: Temps de finestra recomanats en funció de la freqüència de mostreig

Freq. de mostreig (Hz)	Temps de finestra (ms)	mostres per finestra	FFT	Zero padding
48000	85	4080	4096	16
	42	2016	2048	32
	21	1008	1024	16
44100	92	4057	4096	39
	46	2028	2048	20
	23	1014	1024	10
22050	92	2028	2048	20
	46	1014	1024	10
	23	507	512	5
16000	128	2048	2048	0
	64	1024	1024	0
	32	512	512	0

A més, per al càlcul correcte de la FFT, es necessari que el senyal sigui periòdic. Per obtenir certa periodicitat, cada un d'aquests segments es multiplica per una finestra Hamming. Amb MATLAB es pot crear aquesta finestra de *N* mostres amb la funció **hamming**. Aquesta funció aproxima els valors d'inici i final a zero, multiplicant el valor d'amplitud de cada mostra com indica la següent expressió:

$$w[n]=0.54-0.46\cos\left(2\pi\frac{n}{N}\right), \qquad 0\leq n\leq N \tag{9}$$

A continuació es mostra la representació gràfica d'aquesta funció, aplicant una finestra de 512 mostres:

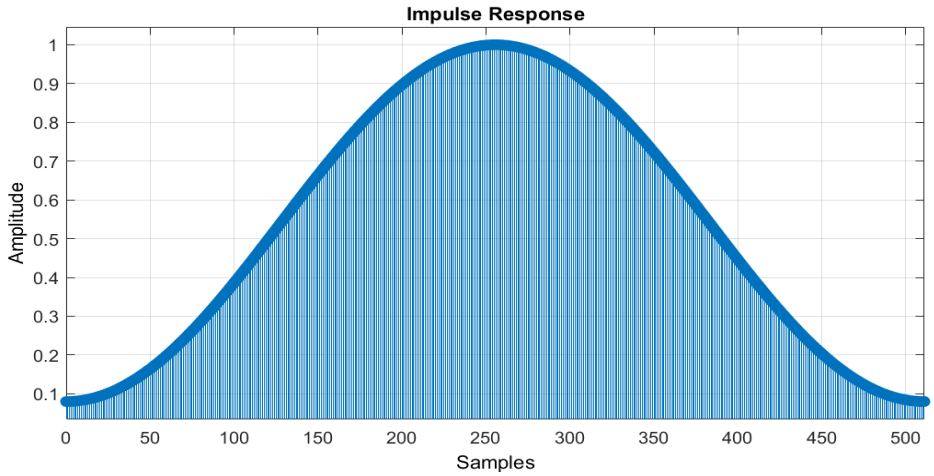

Figura 16: Finestra Hamming

5.2 Extracció de característiques

El submòdul d'extracció de característiques s'executa amb la funció **FEATURES_ EXTRACTION.m**. Aquesta funció rep una matriu amb els segments enfinestrats, procedents del submòdul de preprocessament, i la freqüència de mostreig. Es generen dues sortides:

1. Una estructura amb els valors de cada característiques extreta (en forma de vector) per a cada segment
2. Una estructura amb els paràmetres estadístics d'aquests vectors.

Tot i que la primera sortida no s'utilitza en la resta del projecte, és interessant obtenir-la de cara a futures millores. Per exemple, la classificació amb sistemes que utilitzen les característiques en si i no pas els paràmetres estadístics d'aquestes; com en el cas del sistema de classificació HMM.

Les característiques escollides són:

- *pitch* o freqüència fonamental
- energia en escala logarítmica
- primera derivada del *pitch*
- primera derivada de l'energia
- 12 coeficients LFPC.

De cada una d'aquestes es calculen els 12 paràmetres estadístics. Aquesta combinació és experimental ja que en cap altre treball d'aquesta bibliografia s'ha utilitzat. A més, els estudis que proposen l'extracció dels LFPC realitzen l'entrenament del classificador sobre aquests coeficients directament i no sobre els paràmetres estadístics calculats sobre els coeficients.

5.2.1 Pitch

La freqüència fonamental és la vibració que es genera en els plegaments de la glotis degut a les alenades d'aire en la producció de la parla. Oscil·la, aproximadament, entre els 80 Hz i els 300 Hz. L'efecte de les emocions en a l'aparell fonador pot alterar aquesta característica acústica.

En aquest projecte es fa una estimació del *pitch* mitjançant l'ús del *cepsturm*. El *cepstrum* d'un senyal és el resultat d'aplicar la transformada inversa de Fourier del logaritme de l'espectre del senyal. L'expressió matemàtica és:

$$Cepstrum(w[n]) = F^{-1}(\log(|F(w[n])|)) \tag{10}$$

El primer pas és definir un rang de temps. En aquest projecte el rang va de 2.5 ms a 16 ms, que correspon a un rang de freqüències entre 62.5 Hz fins als 400 Hz. Després, es calcula el *cepstrum* del senyal amb la funció **cceps** i es localitza el pic més alt dins aquest rang, que representa el *pitch.*

Aquest procediment el realitza la funció **pitch.m**, creada seguint l'exemple que ofereix la web *mathworks* (http://es.mathworks.com/help/signal/ug/estimating-fundamental-frequency-with-the-complex-cepstrum.html?searchHighlight=cepstrum).

Amb el vector de *pitch* de cada segment es calcula la primera derivada per tenir una visió del comportament d'aquest a nivell temporal.

Les següents gràfiques mostren la distribució de densitat del *pitch* i la seva derivada per a un conjunt de dades de 4 emocions: enuig, alegria, tristesa i neutre. Processat amb els valors per defecte:

- freqüència de mostreig de 22050 Hz
- freqüència de tall del filtre a 100 Hz
- temps de finestra 23 ms
- temps d'encavalcament 11.5 ms.

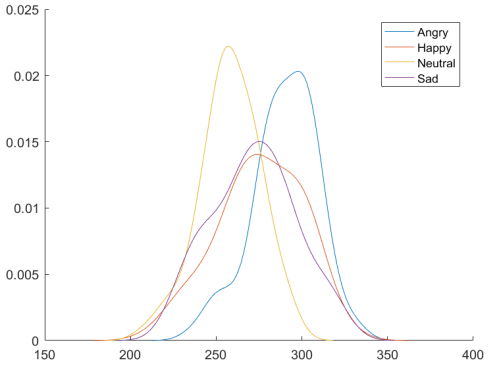

Figura 17: Distribució de densitat: pitch

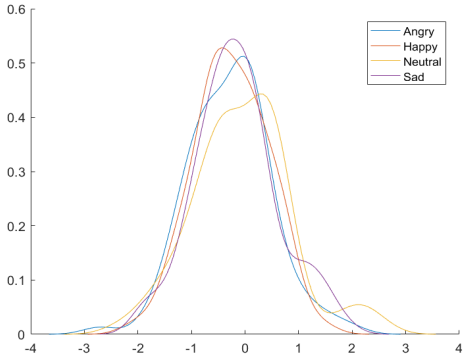

Figura 18: Distribució de densitat: derivada del pitch

5.2.2 Energia

L'energia en escala logarítmica o *log energy* es calcula com el logaritme de la suma del valor de les mostres de cada *frame,* amb l'expressió següent:

$$E = \log \sum_{n=1}^{N} w[n]^2 \tag{11}$$

La funció **log_energy.m**, rep una matriu de segments per mostres (*N)* i calcula l'energia de cada un dels segments enfinestrats (*w[n]*). Aquesta característica pot ser una valoració de la intensitat de la parla, la qual es veu afectada de forma evident pels diferents estats emocionals.

De manera anàloga a com s'ha fet amb el *pitch* i la seva derivada, es presenten les distribucions de densitat de l'energia i la seva derivada.

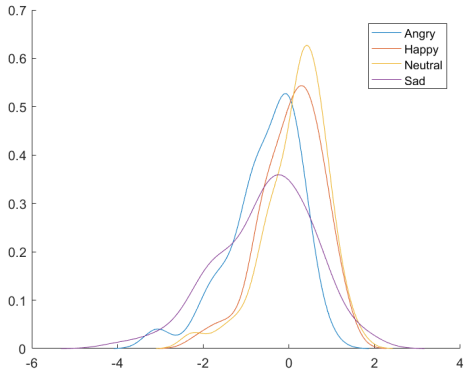

Figura 19: Distribució de densitat: log energy

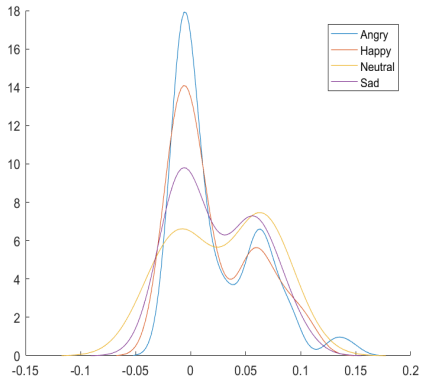

Figura 20: Distribució densitat: derivada log energy

5.2.3 Short time Log Frecuency Power Coefficients (LFPC)

Una mesura del contingut emocional en la parla és la distribució espectral de l'energia en el rang de freqüencial de la parla. Cada subbanda ofereix informació de la distribució energètica i de la intensitat dins del rang de freqüències específic. Per extreure els coeficients LFPC s'han seguit els procediments de [13].

S'ha dissenyat un banc de filtres per dividir el senyal de la parla en 12 bandes de freqüència que coincideixen amb el bandes de percepció crítiques de l'oïda humana. Aquest banc s'implementa a la funció **lfpc_bancFilter.m**, que rep un vector de freqüències (que depèn de la freqüència de mostreig) i 3 paràmetres opcionals:

- amplada del filtre inicial (b_1)
- freqüència central inicial (fc_1)
- factor de creixement logarítmic del filtre (α)

Seguin la metodologia de [13], aquests paràmetres per defecte són b_1=54 Hz, fc_1=127Hz i α=1.4 (ja que és el valor que ofereix millors resultats). El procediment matemàtic per a al desenvolupament del banc és:

$$b_1 = C \tag{12}$$

$$b_i = \alpha\, b_{i-1}, \qquad 2 \le i \le 12 \tag{13}$$

$$fc_i = f_1 + \sum_{j=1}^{i-1} b_j + \frac{b_i - b_1}{2} \tag{14}$$

on, fc_i i b_i són la freqüència central i l'ample de banda de cada filtre, respectivament. L'amplada de banda inicial és $b_1 = C$= 57 Hz i la freqüència inicial és f_1=100 Hz. La taula 3 mostra la configuració del banc de filtres per als valors per defecte d'aquest projecte.

Taula 3: Banc de filtres LFPC

Filtre *i*	α=1.4	
	fc_i	b_i
1	127	54
2	192	75
3	281	104
4	406	145
5	579	202
6	820	280
7	1155	389
8	1620	541
9	2267	753
10	3167	1046
11	4417	1454
12	6154	2021

Amb aquest banc de filtres el rang freqüencial va de 100 Hz fins als 7165 Hz, ocupant així l'expecte més característic de la parla.

Un cop dissenyat el banc de filtres es computen els coeficients LFPC amb la funció **lfpc.m**, que rep la matriu de segments enfinestrats i la freqüència de mostreig. Cada segment es transforma al domini freqüencial amb la FFT. La resposta de la FFT de cada segment a cada filtre és la distribució espectral energètica compresa entre els marges del filtre, que correspon a la resposta d'una finestra rectangular $W_m(k)$:

$$W_m(k)=\begin{cases}1 & l_m \le k \le h_m \\ 0 & others\end{cases}, \qquad \text{m=1,2,...,12} \tag{15}$$

on, l_m i h_m són els extrems baix i alt del filtre m i k el domini de la FFT. La figura 21 mostra l'aplicació del banc de filtres sobre un segment aleatori d'un dels àudios de la base de dades.

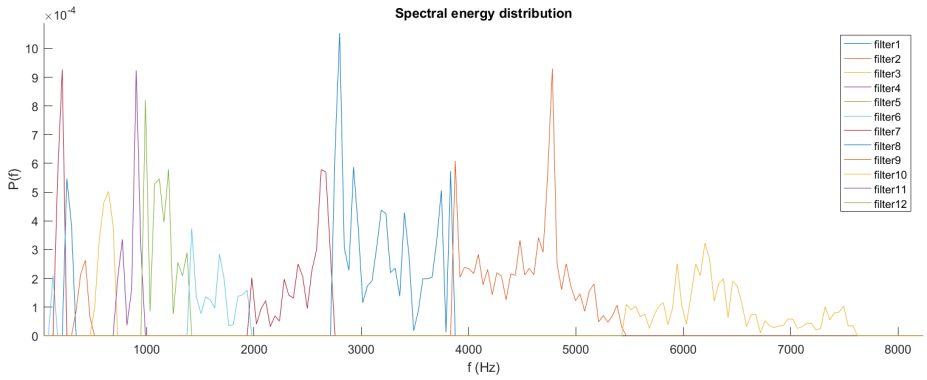

Figura 21: Distribució espectral d'energia

La sortida a cada filtre ve donada per:

$$S_t(m)=\sum_{k=l_m}^{h_m}\left(X_t(k)W_m(k)\right)^{2,} \qquad m=1,2,\ldots,12 \tag{16}$$

on, m indica el filtre, t el segment, $X_t(k)$ es el k-èssim component espectral del senyal enfinestrat.

Finalment, s'obtenen els 12 coeficients LFPC per cada segment amb la següent expressió:

$$LFPC_t(m)=\frac{10\log_{10}(S_t(m))}{N_m}, \qquad m=1,2,\ldots,12 \tag{17}$$

on, N_m és el nombre de components espectrals del filtre m.

Igual que amb a les altres característiques, es mostra la distribució de densitat de cada coeficient LFPC.

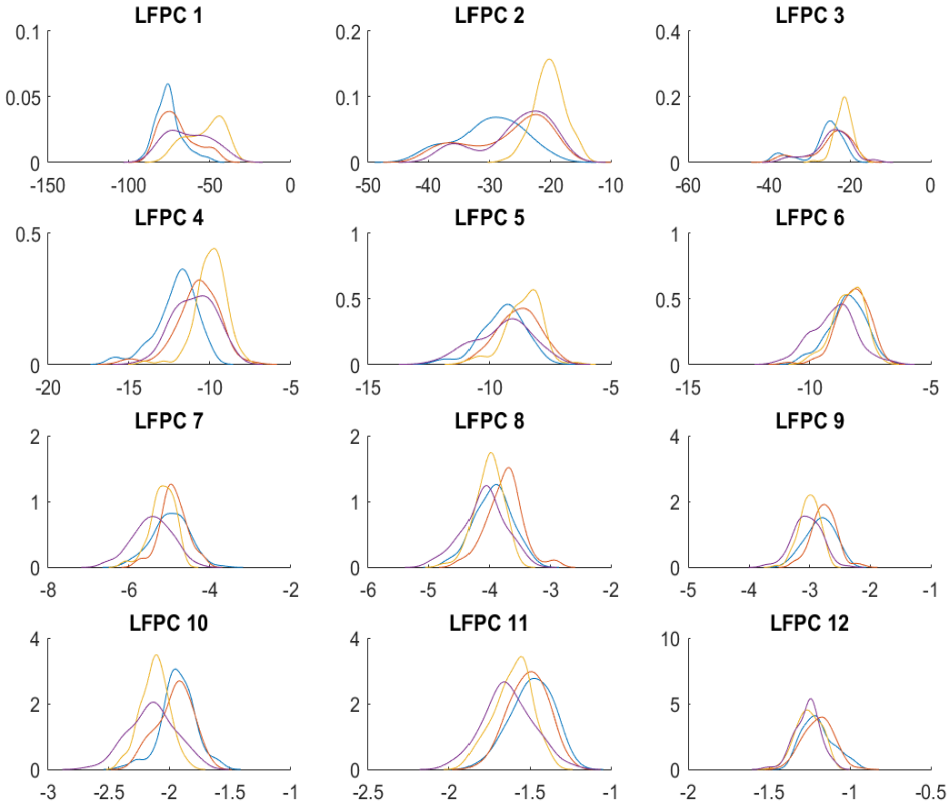

Figura 22: Distribució de densitat: LFPC

5.2.4 Paràmetres estadístics

De cada una d'aquestes característiques s'obté un vector amb els valors de cada segment. D'aquests vectors se'n calculen 12 paràmetres estadístics amb la funció **aux_stats.m**, que s'executa dins la funció **calc_statistics.m** del submòdul d'extracció de característiques. Aquests paràmetres s'enumeren a continuació:

1. **Mitjana (*mean*)**

   La mitjana s'obté sumant els valors de totes les dades d'un conjunt i dividint-lo pel nombre d'elements del conjunt. Per a un conjunt de dades A amb N mostres, la mitjana es calcula com:

$$\mu = \frac{1}{N}\sum_{i=1}^{N} A_i \tag{18}$$

2. **Valor mínim (*min*)**

   El valor més petit del conjunt.

3. **Valor màxim (*max*)**

   Valor més gran del conjunt.

4. **Desviació estàndard (*std*)**

   Indica el grau de variació respecte a la mitjana. Un valor baix indica que els valor tendeixen a estar pròxims a la mitjana, mentre que un valor alt indica que els valors estan dispersos al llarg de tot el rang. Per a un conjunt de dades *A* amb *N* mostres, es calcula com:

$$S=\sqrt{\frac{1}{N-1}\sum_{i=1}^{N}\left|(A_i-\mu)\right|^2} \tag{19}$$

5. **Asimetria (*skewness*)**

   Skewness és una mesura de la asimetria de les dades al voltant de la mitjana de la mostra. Si l'asimetria és negativa, les dades s'estenen més cap a l'esquerra de la mitjana que cap a la dreta. Si l'asimetria és positiva, les dades s'estenen més cap a la dreta. L'asimetria de la distribució normal (o qualsevol distribució perfectament simètrica) és zero. Es defineix com:

$$S=\frac{E(x-\mu)^3}{\sigma^3} \tag{20}$$

   on, *E(t)* és l'esperança de *t*, μ és la mitjana de *x* i σ és al desviació estàndard de *x*.

6. **Curtosi (*kurtosis*)**

   La curtosi és el grau d'apuntament d'una distribució estadística. Representa la convexitat d'una funció de distribució. En una distribució normal el valor de curtosi és 3. Es defineix com:

$$K=\frac{E(x-\mu)^4}{\sigma^4} \tag{21}$$

   on, *E(t)* és l'esperança de *t*, μ és la mitjana de *x* i σ és al desviació estàndard de *x*.

7. **Quantils (*quantile;* Q1, Q2 i Q3)**

   És una mesura de posició no central que permet conèixer altres punts característics de la distribució que no són els valors centrals. El quantil d'ordre p d'una distribució (amb $0 < p < 1$) és el valor de la variable x_p que marca un tall de manera que una proporció p de valors de la població és menor o igual que x_p. Per exemple, el quantil d'ordre 0.36 deixaria un 36% de valors per sota i el quantil d'ordre 0.50 es correspon amb la mediana de la distribució.

   Els quantils solen usar-se per grups que divideixen la distribució en parts iguals; enteses aquestes com a intervals que comprenen la mateixa proporció de valors. En aquest projecte es fan servir els quantils 0.25, 0.50 i 0.75 de la distribució normal; coneguts com els quartils *Q1*, *Q2* i *Q3,* que divideixen la distribució en quatre blocs, cadascun dels quals conté el 25% de les dades.

8. **Rang interquartílic (*interquertile range; iqr*)**

   El rang interquartílic es defineix com la diferència entre el quartil tercer i el quartil primer. En aquest rang hi ha, per la mateixa definició dels quartils, el 50% de les observacions.

9. **Desviació semiinterquartílica (*semi-interquartile deviation; SID*)**

   La SID és defineix com la meitat de la diferència entre els quartils primer i tercer. Aquesta diferència és per definició el rang interquartílic. Per tant, és pot calcular com *iqr/2*.

10. **Percentil (*percentile, prctile*)**

    El percentil és una mesura de tendència central que indica, un cop ordenades les dades de menor a major, el valor de la variable per sota del qual es troba un percentatge donat de dades en un grup d'observacions. Per exemple, el percentil 20 és el valor sota el qual es troben el 20% de les observacions. Ena quest projecte s'utilitza el 90%.

5.3 Selecció de característiques (postprocessament)

Aquest submòdul s'implementa amb la funció **FEATURES_SELECTION.m**. Rep a l'entrada una taula amb totes les estadístiques de cada àudio i genera una llista amb les característiques ordenades de major a menor rellevància segons el criteri d'informació mútua.

En aquest bloc, com s'ha comentat, no s'inclou l'estandardització de les dades dins el mateix rang de magnitud, degut a que el sistema de classificació SVM permet fer-ho de manera automàtica.

5.3.1 Informació mútua

En la teoria de la probabilitat, i la teoria de la informació, la informació mútua de dues variables aleatòries és una quantitat que mesura la dependència mútua de dues variables, és a dir, mesura la reducció de la incertesa (entropia) d'una variable, *X*, a causa del coneixement del valor d'una altra variable *Y.* La informació mútua $I(x_i ; y_j)$, es defineix com:

$$I(x_i; y_i) = \log \frac{P(x_i | y_i)}{P(x_i)} \tag{22}$$

Aquest procés s'implementa amb la funció **MI.m** (Okko Rasanen, 2013), que rep una matriu de dades (observacions per característiques) i un vector d'etiquetes o classes que correspon a cada observació. La sortida és un vector, amb la posició de les característiques de la matriu donada, ordenat per rellevància i, un vector amb els pesos que li correspon a cada posició.

La taula 4 mostra la llista de característiques i els pesos corresponents, extretes amb aquesta funció, per un conjunt de dades de 4 emocions: enuig, alegria, tristesa i neutre. Processat amb els valors per defecte:

- freqüència de mostreig de 22050 Hz
- freqüència de tall del filtre a 100 Hz

- temps de finestra 23 ms
- temps d'encavalcament 11.5 ms.

Taula 4: Paràmetres estadístics per ordre de rellevància

	w	feature		w	feature		w	feature		w	feature		w	feature		w	feature
1	0,5085	'LFPC2_Q3'	33	0,3349	'LFPC9_Q2'	65	0,2390	'LFPC11_Q2'	97	0,1837	'LFPC2_KURT'	129	0,1470	'LFPC1_KURT'	161	0,1051	'dlogE_MIN'
2	0,5046	'LFPC12_SKEW'	34	0,3291	'dF0_Q1'	66	0,2373	'LFPC3_MAX'	98	0,1828	'LFPC4_IQR'	130	0,1456	'logE_MAX'	162	0,1051	'F0_MIN'
3	0,4740	'LFPC2_P90'	35	0,3258	'LFPC11_SKEW'	67	0,2317	'LFPC3_P90'	99	0,1828	'LFPC4_SID'	131	0,1454	'LFPC2_STD'	163	0,1046	'LFPC6_KURT'
4	0,4652	'LFPC2_Q2'	36	0,3232	'LFPC6_Q3'	68	0,2307	'LFPC6_MIN'	100	0,1821	'LFPC8_Q1'	132	0,1426	'logE_STD'	164	0,1021	'LFPC5_Q3'
5	0,4168	'LFPC10_Q3'	37	0,3219	'F0_Q1'	69	0,2305	'LFPC8_Q3'	101	0,1817	'LFPC5_MIN'	133	0,1422	'LFPC2_MIN'	165	0,0993	'dlogE_MAX'
6	0,4099	'LFPC1_Q2'	38	0,3193	'LFPC10_MEAN'	70	0,2302	'LFPC12_Q1'	102	0,1800	'LFPC6_MEAN'	134	0,1413	'LFPC11_P90'	166	0,0990	'LFPC3_KURT'
7	0,4065	'LFPC1_P90'	39	0,3153	'LFPC3_Q3'	71	0,2295	'LFPC3_Q2'	103	0,1733	'LFPC1_IQR'	135	0,1406	'dF0_MAX'	167	0,0987	'F0_Q3'
8	0,4049	'LFPC2_MEAN'	40	0,3041	'LFPC7_MAX'	72	0,2295	'LFPC3_Q1'	104	0,1733	'LFPC1_SID'	136	0,1365	'LFPC10_STD'	168	0,0977	'LFPC4_KURT'
9	0,4046	'LFPC2_MAX'	41	0,3041	'F0_STD'	73	0,2291	'LFPC9_Q1'	105	0,1719	'dlogE_SKEW'	137	0,1356	'LFPC11_MIN'	169	0,0958	'LFPC9_SKEW'
10	0,4008	'LFPC1_MEAN'	42	0,3014	'LFPC4_P90'	74	0,2230	'LFPC8_P90'	106	0,1713	'LFPC4_MIN'	138	0,1258	'LFPC11_STD'	170	0,0948	'dlogE_KURT'
11	0,4001	'LFPC10_P90'	43	0,2985	'LFPC10_MAX'	75	0,2210	'LFPC3_MEAN'	107	0,1713	'LFPC7_SKEW'	139	0,1255	'dlogE_Q1'	171	0,0946	'LFPC8_SKEW'
12	0,3995	'LFPC1_MAX'	44	0,2982	'LFPC12_Q2'	76	0,2166	'LFPC6_STD'	108	0,1667	'LFPC2_IQR'	140	0,1248	'LFPC12_MAX'	172	0,0931	'dlogE_MEAN'
13	0,3977	'LFPC4_Q2'	45	0,2915	'dF0_KURT'	77	0,2152	'logE_IQR'	109	0,1667	'LFPC2_SID'	141	0,1234	'LFPC5_STD'	173	0,0929	'dF0_MEAN'
14	0,3937	'LFPC9_P90'	46	0,2902	'LFPC11_Q1'	78	0,2152	'logE_SID'	110	0,1665	'LFPC6_MAX'	142	0,1227	'LFPC10_IQR'	174	0,0916	'LFPC10_KURT'
15	0,3908	'LFPC7_P90'	47	0,2890	'LFPC10_Q1'	79	0,2143	'LFPC4_MAX'	111	0,1661	'LFPC8_MEAN'	143	0,1227	'LFPC10_SID'	175	0,0902	'LFPC5_MAX'
16	0,3864	'LFPC1_Q3'	48	0,2836	'LFPC7_Q2'	80	0,2122	'LFPC7_Q1'	112	0,1657	'dlogE_IQR'	144	0,1211	'LFPC11_MAX'	176	0,0891	'logE_Q3'
17	0,3787	'LFPC9_MAX'	49	0,2813	'LFPC12_MIN'	81	0,2113	'dlogE_Q3'	113	0,1657	'dlogE_SID'	145	0,1193	'LFPC5_KURT'	177	0,0886	'logE_P90'
18	0,3748	'LFPC2_Q1'	50	0,2808	'logE_MIN'	82	0,2113	'LFPC9_IQR'	114	0,1641	'LFPC1_STD'	146	0,1184	'LFPC12_STD'	178	0,0859	'LFPC8_STD'
19	0,3738	'LFPC4_Q3'	51	0,2799	'LFPC2_SKEW'	83	0,2113	'LFPC9_SID'	115	0,1635	'LFPC8_IQR'	147	0,1162	'LFPC5_P90'	179	0,0836	'dlogE_P90'
20	0,3711	'LFPC6_IQR'	52	0,2791	'F0_MEAN'	84	0,2080	'LFPC5_MEAN'	116	0,1635	'LFPC8_SID'	148	0,1155	'LFPC4_STD'	180	0,0823	'LFPC12_P90'
21	0,3711	'LFPC6_SID'	53	0,2740	'LFPC8_MAX'	85	0,2047	'LFPC11_MEAN'	117	0,1634	'LFPC6_Q2'	149	0,1134	'LFPC4_SKEW'	181	0,0822	'dF0_SKEW'
22	0,3668	'LFPC7_Q3'	54	0,2713	'LFPC9_MEAN'	86	0,2040	'LFPC5_IQR'	118	0,1632	'LFPC8_MIN'	150	0,1128	'LFPC3_MIN'	182	0,0808	'dlogE_STD'
23	0,3649	'LFPC10_Q2'	55	0,2667	'LFPC12_Q3'	87	0,2040	'LFPC5_SID'	119	0,1611	'LFPC11_KURT'	151	0,1126	'LFPC6_SKEW'	183	0,0804	'logE_KURT'
24	0,3540	'LFPC4_MEAN'	56	0,2664	'F0_IQR'	88	0,2021	'LFPC11_Q3'	120	0,1606	'LFPC5_Q2'	152	0,1114	'logE_SKEW'	184	0,0796	'LFPC7_KURT'
25	0,3530	'dF0_STD'	57	0,2664	'F0_SID'	89	0,1973	'LFPC9_STD'	121	0,1602	'LFPC10_MIN'	153	0,1103	'LFPC11_IQR'	185	0,0791	'LFPC12_KURT'
26	0,3521	'LFPC1_Q1'	58	0,2618	'F0_KURT'	90	0,1969	'LFPC8_Q2'	122	0,1597	'logE_MEAN'	154	0,1103	'LFPC11_SID'	186	0,0774	'LFPC9_KURT'
27	0,3481	'LFPC4_Q1'	59	0,2560	'LFPC7_MEAN'	91	0,1921	'LFPC6_Q1'	123	0,1578	'LFPC1_MIN'	155	0,1102	'LFPC3_IQR'	187	0,0754	'LFPC12_IQR'
28	0,3457	'dF0_IQR'	60	0,2506	'LFPC5_Q1'	92	0,1905	'LFPC7_STD'	124	0,1568	'F0_SKEW'	156	0,1102	'LFPC3_SID'	188	0,0754	'LFPC12_SID'
29	0,3457	'dF0_SID'	61	0,2493	'LFPC7_IQR'	93	0,1898	'LFPC7_MIN'	125	0,1565	'LFPC9_MIN'	157	0,1072	'LFPC3_SKEW'	189	0,0741	'dF0_Q2'
30	0,3457	'dF0_P90'	62	0,2493	'LFPC7_SID'	94	0,1895	'LFPC1_SKEW'	126	0,1557	'LFPC10_SKEW'	158	0,1071	'LFPC8_KURT'	190	0,0708	'LFPC5_SKEW'
31	0,3441	'LFPC9_Q3'	63	0,2468	'logE_Q1'	95	0,1883	'LFPC12_MEAN'	127	0,1523	'dF0_MIN'	159	0,1059	'F0_P90'	191	0,0623	'dlogE_Q2'
32	0,3367	'dF0_Q3'	64	0,2409	'LFPC6_P90'	96	0,1878	'logE_Q2'	128	0,1474	'F0_Q2'	160	0,1054	'LFPC3_STD'	192	0,0000	'F0_MAX'

Capítol 6 Implementació: Mòdul de classificació

Aquest mòdul genera, durant la fase d'entrenament, el model que serveix per a realitzar les perdicions sobre dades desconegudes. S'implementa amb la funció **CLASSIFICATION_MODULE.m**, els paràmetres d'entrada de la qual són:

1. La taula de observacions per paràmetres estadístics provinent del modul d'extracció.
2. La llista de característiques ordenades per rellevància provinent del modul d'extracció.
3. Les etiquetes de cada classe d'observació, creades en el procès de càrrega de dades.

Durant aquest procés s'entrenen tants models com característiques hi ha a la llista; això implica un total de 192 entrenaments. El primer entrenament es realitza amb la primera característica de la llista (la més rellevant) i es va afegint, de manera seqüencial, una nova característica per a cada nou entrenament. Per a cada model nou es compara la seva precisió amb la precisió del model anterior. Així, a la sortida s'obté el millor model dels 192 i el nombre de característiques utilitzades. Per exemple, per a un sistema entrenat amb una base de dades de 4 emocions bàsiques (enuig, alegria, tristesa i neutre), de 96 frases per cada estat, processat amb els valors per defecte (freqüència de mostreig de 22050 Hz, freqüència de tall del filtre a 100 Hz, temps de finestra 23 ms i temps d'encavalcament 11.5 ms), s'obté un model amb una precisió del 88% amb 132 característiques. Això representa una reducció del 31.25% del total dels paràmetres.

La raó per la qual s'ha utilitzat aquesta metodologia és per mantenir l'estructura modular dissenyada i oferir un cert grau d'escalabilitat. Altres mètodes de selecció de característiques s'implementen sobre un classificador en concret i, per tant, no permeten separar els processos de selecció i classificació. Amb aquest disseny es poden afegir nous mètodes de selecció de característiques i diferents sistemes de classificació.

En aquest procés també es validen els resultats i s'obté la validació de les perdicions a la sortida, que serveix per crear la matriu de confusió.

6.1 Sistema de classificació SVM

Per tal d'escollir un sistema de classificació s'han realitzat diferents experiments amb diferents sistemes gracies a l'aplicació de MATLAB: *Classification Learner app*. Aquesta, permet de forma molt intuïtiva estudiar diferents classificadors i veure'n els resultats, a més a més d'exportar el codi del classificador seleccionat. Com era d'esperar després de l'estudi previ, el que millors resultats ha donat és el SVM amb una funció de *kernel* quadràtica i el mètode multi-classe *one-vs-all* («un contra tots»).

Una màquina de vectors és un sistema que pren un conjunt de dades a l'entrada i prediu -a la sortida- , per cadascuna d'aquestes entrades, a quina de les possibles classes pertany. Mitjançant l'entrenament amb dades d'entrada prèviament etiquetades, s'estableix un model que separa les classes d'entrada. Aquest model *N*-dimensional estableix una frontera entre les tipologies establertes, aquesta se situa en el punt en el qual la diferència entre classes sigui el més gran possible i el marge d'error sigui zero (conjunt separable) o mínim (conjunt no separable). S'anomenen vectors de suport als punts que conformen les dues línies paral·leles a l'hiperplà, essent aquesta distància la major possible (marge).

Tot i que, en un principi, el problema ha estat declarat en un espai de dimensions finites, és possible que el conjunt de dades en aquest espai no sigui separable linealment. És per aquesta raó que l'espai es mapa a un espai de dimensions majors per ajudar a fer que sigui separable en el nou espai. Per mantenir una càrrega computacional raonable els esquemes de mapatge de les màquines SVM són dissenyats per a assegurar que els productes vectorials es puguin calcular en els termes de l'espai original, definint-los en

termes d'una funció anomenada funció de *kernel* (nucli del SVM) adaptada per a cada tipus de problema. La funció *kernel* d'aquest projecte és una funció polinòmica d'ordre p=2:

$$k(x_1, x_2) = (x_1^T x_2 + 1)^p \tag{23}$$

Per a fer un entrenament multi classe (*one-vs-all*) s'entrena un classificador per cada classe, agafant les mostres de la resta de classes (en un conjunt) coma negatives; de manera que s'obtenen dues tipologies (la classe actual contra la resta). Per a la perdició s'apliquen tots els classificadors i s'obté, de cada un, el valor de classificació normalitzat. El resultat és la classe que obté major resultat normalitzat.

Amb MATLAB aquest procediment s'executa amb la funció **fitcecoc**:

```
classificationSVM=fitcecoc(predictors,response,'Learners',template,'Coding','onevsall','ClassNames',labels);
```

on, *predictors* fa referència a les característiques, *response* són les classes, *template* és la plantilla que conté tots les opcions del classificador com, per exemple, el tipus de funció *kernel* o l'opció d'estandardització de les dades.

6.2 Validació creuada

Un cop creat el classificador, aquest s'ha de validar. El mètode per fer aquesta comprovació és el *k-fold cross validation*; en aquest projecte per defecte k=10.

El procediment és el següent: les dades de mostra es divideixen en k subconjunts. Un dels subconjunts s'utilitza com a dades de prova i la resta (k-1) com a dades d'entrenament. El procés de validació creuada es va repetint durant les k iteracions, amb cada un dels possibles subconjunts de dades de prova. Finalment es realitza la mitjana aritmètica dels resultats de cada iteració per obtenir un únic resultat. Aquest mètode és molt precís ja que s'avaluen a partir de k combinacions de dades d'entrenament.

Aquest procediment es du a terme amb la funció **crossval**, el qual genera un model particionat en k conjunts. D'aquest, es pot calcular la precisió (**1-kfoldLoss**) i les prediccions validades (**kfoldPredict**) amb les quals s'obté la matriu de confusió.

Tant el sistema SVM com la validació creuada s'executen dins al funció **trainClassifier.m,** la qual s'ha generat amb l'aplicació *Classification Learner app.* Per encabir-la dins del projecte, s'ha modificat els paràmetres d'entrada i sortida. A l'entrada s'han afegit les següents variables:

1. Un vector amb el nom de cada característica que entrena el model, les quals s'afegeixen de forma seqüencial per a cada entrenament.
2. Un vector que conté el nom de les etiquetes, atès que el sistema pot entrenar diferents conjunts de dades amb diferents etiquetes.

S'ha modificat la funció per a oferir, a més a més del model, les següents sortides:

1. El valor de precisió
2. La validació de les prediccions.

Capítol 7 Interfície gràfica: CTSSER GUI

La interfície gràfica permet usar les funcions implementades de manera intuïtiva i transparent per a l'usuari. Permet introduir els paràmetres opcionals recomanats i observar els resultats d'una manera senzilla i pràctica.

7.1 Implementació

Per a la implementació s'ha usat l'entorn de desenvolupament GUIDE que proporciona MATLAB i, s'ha creat i modificat la funció **CTSSER_GUI.m** per a realitzar tots els procediments necessaris.

Seguint l'esquema de fases d'execució d'entrenament i predicció, quan s'executa la GUI, es mostren dos panells principals: *Train classifier* i *Prediction*. Bàsicament, en el primer cas es fa ús de la funció d'entrenament **SER_TRAINING.m**, i en el segon la funció **SER_PREDICT.m**.

Quan s'executa la GUI es crea un directori nou (dins dels directori actual) anomenat CTSSER_Models on es guardaran, en subdirectoris independents, els models creats juntament amb la matriu de confusió generada. Aquests subdirectoris es creen de forma automàtica amb el format de nom següent:

{Nom directori principal}_{freq. rate}_{frame size}_{HPF cutoff}

Una de les propietats d'aquesta GUI és la finestra que mostra l'execució del programa de forma anàloga a la funció SER_TRAINING.m, que mostra el progrés en la finestra de comandaments de MATLAB. Per tal d'imprimir els resultats a la pròpia GUI, s'ha creat la funció auxiliar **CTS_listboxDraw.m**. Aquesta funció llegeix el contingut del fitxer **log.txt** (que es crea al iniciar un entrenament) i el mostra a la GUI. Prèviament, s'ha escrit en aquest fitxer passant el seu identificador com a paràmetre a la resta de funcions

que executen la funció *fprintf*. Aquest identificador és el paràmetre opcional **fid** mentre que la referència a la pantalla de sortida de la GUI és el paràmetre opcional **CTSout** (veure apartat 4.1, «Fase d'entrenament»).

7.2 Ús de la interfície gràfica

La figura 23 mostra l'aparença de la interfície gràfica. S'han emmarcat i enumerat les diferents opcions i funcions.

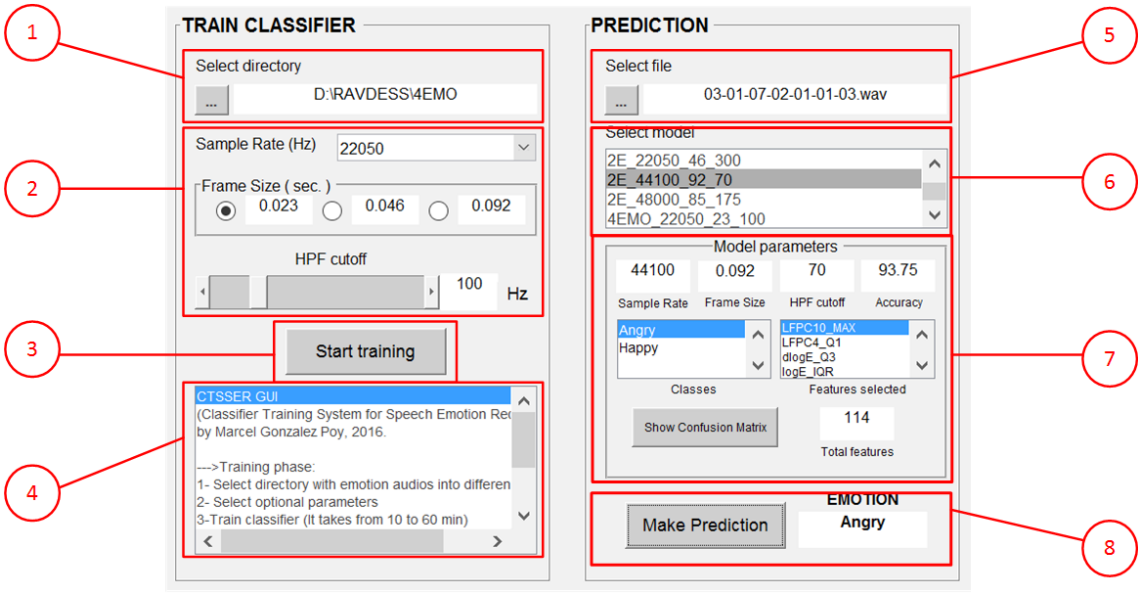

Figura 23: Interfície gràfica CTSSER GUI

A continuació s'expliquen les funcionalitats que ofereix:

1. **Selecció de directori**

   Permet seleccionar, mitjançant un quadre de diàleg, el directori principal on hi ha els àudios de veus emocionades separats per carpetes independents.

2. **Introducció de paràmetres opcionals**

   a. Freqüència de mostreig: mitjançant un *popup menu* es pot seleccionar una freqüència de mostreig d'entre les recomanades: 16000, 22050, 44100, 48000 Hz. Per defecte el valor inicial és 22050 Hz.

   b. Temps de finestra: es mostren els tres valors de temps recomanats segons la freqüència de mostreig seleccionada (veure taula 2).

c. Freqüència de tall del filtre passa alt: mitjançant un *slider* permet seleccionar la freqüència de tall del filtre entre 50 i 300 Hz.

3. **Començar entrenament**

   Un cop escollits els paràmetres, la interfície permet clicar el botó per començar l'entrenament del classificador. Es crea el fitxer **log.txt** i es mostra el progrés per la finestra de text de la GUI (punt 4).

4. **Finestra de text**

   Al inici, aquest quadre mostra les instruccions resumides del funcionament de la GUI. Un cop comença l'entrenament es mostra el contingut actualitzat del fitxer log.txt.

5. **Selecció de fitxer d'àudio**

   Permet seleccionar un fitxer *wav* mitjançant un quadre de diàleg. Sobre aquest fitxer es farà la predicció.

6. **Selecció de model**

   Aquest *listbox* mostra els models disponibles que hi ha al directori *CTSSER_Models*.

7. **Paràmetres del model**

   Dins d'aquest panell es mostren els paràmetres del model escollit:

   a. freqüència de mostreig

   b. temps de finestra

   c. freqüència de tall del filtre

   d. percentatge de precisió del model

   e. llistat de classes sobre el que es farà la predicció

   f. llistat de característiques seleccionades durant l'entrenament

   g. nombre total de característiques

h. botó *show confusion matrix*. Mostra la matriu de confusió del model en una nova figura.

8. **Predicció**

Quan s'ha seleccionat un fitxer i un model, la interfície permet clicar sobre el botó *make prediction* per dur a terme la predicció. El resultat es mostra en el quadre *EMOTION*, amb el nom de la classe

Capítol 8 Proves i resultats

En aquest capítol es documenten els resultats de les proves més significatives. Tots els experiments s'han dut a terme amb els àudios de la base de dades RAVDESS, enregistrats amb una freqüència de mostreig de 48000 Hz i 16 bits de quantificació.

Tot i que els resultats aporten una visió general del comportament del sistema, estan condicionats a la base de dades utilitzada. És a dir: l'idioma anglès, els actors (gènere, edat, fisiologia,...), qualitat i rigorositat de l'enregistrament, equips emprats,...

8.1 Prova 1

En aquesta prova s'han obtingut 12 models en funció de la freqüència de mostreig i el temps de finestra recomanats (veure taula 2). Les emocions escollides són: enuig, alegria, tristesa i neutral. De cada una s'han processat 96 frases; per tant, el total d'observacions és de 384. De cada observació s'extreuen 192 paràmetres; el total de dades pera realitzar l'entrenament ascendeix a 73728. Les següents taules mostren els resultats obtinguts:

Taula 5: Precisió del classificador per a una freqüència de mostreig de 16000 Hz

Freqüència de Mostreig (Hz)	**16000**		
Temps de finestra (ms)	**128**	**64**	**32**
Precisió del model	83.33%	84.90%	83.59%
Nombre de paràmetres	122	97	126

Taula 6: Precisió del classificador per a una freqüència de mostreig de 22050 Hz

Freqüència de Mostreig (Hz)	**22050**		
Temps de finestra (ms)	**92**	**46**	**23**
Precisió del model	84.90%	85.42%	88.02%
Nombre de paràmetres	119	129	131

Taula 7: Precisió del classificador per a una freqüència de mostreig de 44100 Hz

Freqüència de Mostreig (Hz)	**44100**		
Temps de finestra (ms)	**92**	**46**	**23**
Precisió del model	87.24%	85.94%	86.46%
Nombre de paràmetres	83	125	147

Taula 8: Precisió del classificador per a una freqüència de mostreig de 48000 Hz

Freqüència de Mostreig (Hz)	**48000**		
Temps de finestra (ms)	**85**	**42**	**21**
Precisió del model	85.42%	86.20%	88.80%
Nombre de paràmetres	68	138	106

Tot i que aquests resultats poden variar unes dècimes entre diferents experiments, mostren el comportament del sistema de manera general per a diferents configuracions.

Dels resultats se'n deriven les següents conclusions:

- Els pitjors resultats, per sota del 85%, es donen per a la freqüència de mostreig de 16000 Hz. Això pot ser degut a la pèrdua d'informació a causa del remostreig, amb el qual es perden 2 de cada 3 mostres del senyal original. A més aquesta freqüència de mostreig solament permet mostrejar freqüències fins a 8000 Hz , amb la qual cosa es perd la informació per sobre d'aquest rang.

- A partir de 22050 Hz els resultats superen - o s'apropen molt - el 85% de precisió. Es pot assegurar que la informació per sobre dels 8000 Hz és important per a la identificació d'emocions en la parla.
- Per altra banda, els resultats que arriben fins al 88% de precisió es troben en els temps de finestra més petits, és a dir quan hi ha molts segments amb poques mostres per segment. Això vol dir que la informació temporal és important. També implica que la informació més rellevant per a la identificació d'emocions no es troba en les freqüències molt baixes, ja que els temps de finestra petits degraden la informació en freqüències molt baixes. En cas contrari, s'obtindrien millors resultats en temps de finestra elevats.
- Aquests resultats superen els resultats d'alguns dels treballs estudiats. Val a dir que pot estar condicionat a la base de dades utilitzada. Per a fer una comparació real, s'hauria d'utilitzar la mateixa base de dades dels altres projectes.

Les següents figures són les matrius de confusió dels dos classificadors amb millors resultats (48000 Hz – 21 ms i 22050 Hz – 23 ms):

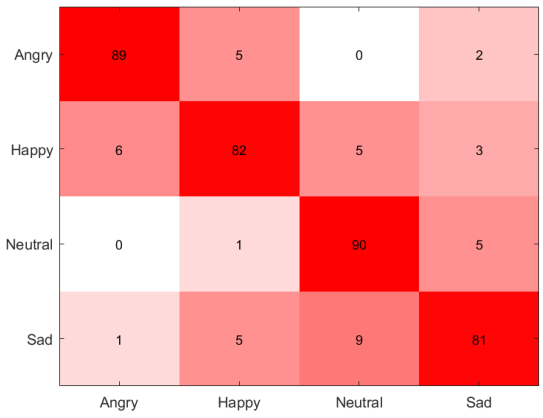

Figura 24: Matriu de confusió (22050Hz -23 ms)

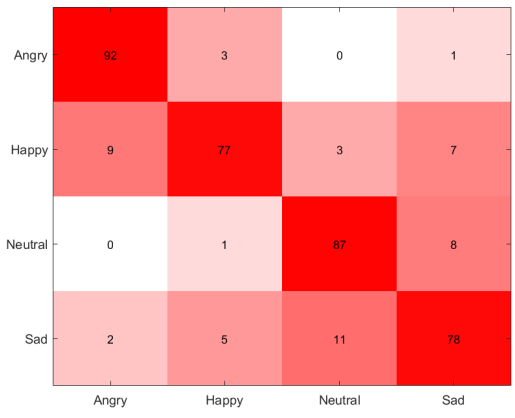

Figura 25: Matriu de confusió (48000Hz - 21 ms)

- D'aquestes matrius es desprèn que on hi ha més problemes d'identificació és entre les emocions amb nivells d'excitació similars: enuig-alegria i neutre-trist. En cas contrari, entre les emocions amb excitacions diferents : enuig-neutre i enuig-trist la confusió és nul·la o gairebé nul·la.

8.2 Prova 2

Aquest experiment mostra l'influencia del filtre passa alt sobre la precisió del classificador. S'han entrenat 5 models amb diferents freqüències de tall del filtre però, tots ells amb una freqüència de mostreig de 22050 Hz i un temps de finestra de 23 ms. Els resultats es mostren en la següent taula:

Taula 9: Precisió en funció del filtre HPF

Freqüència de Mostreig (Hz)	**22050**			
Temps de finestra (ms)	**23**			
Freqüència de tall HPF (Hz)	**50**	**200**	**500**	**1500**
Precisió del model	86.46%	85.42%	80.99%	76,52%
Nombre de paràmetres	82	117	170	148

Com s'ha dit, el filtre s'utilitza per eliminar aquelles freqüències provocades pel soroll que podrien confondre el procés d'entrenament. Així, els valors de tall habituals estan al voltant dels 100 Hz. Ara bé, per veure quin és l'efecte sobre la precisió, en aquesta prova s'han escollit valors extremats.

Es dedueixen les següents conclusions:

- Amb un tall de 50 Hz l'efecte del filtre (comparant amb el de 100 Hz) produeix una petita diferència. Pot ser deguda a que part d'aquestes freqüències, com s'ha dit, provoca un grau de confusió alhora d'entrenar el model. Depenent de la qualitat de l'enregistrament les freqüències per sota dels 100 Hz podrien provocar més problemes.
- A mesura que el tall és més elevat és perd informació. És fa evident a partir de 500 Hz i més a partir de 1500 Hz. Tot i això, el sistema encara obté resultats notables. Aquest fet concorda amb les conclusions anteriors i evidencia que a freqüències mitges-altes hi ha molta informació respecte a l'estat emocional. Val a dir que entre 100 i 200 Hz també hi ha informació útil ja que quan el tall és a 200 Hz és perd precisió.

8.3 Prova 3

Per comprovar la versatilitat del sistema respecte a la capacitat d'entrenar classificadors amb diferents conjunts de dades, s'ha creat un model amb les 7 emocions (exceptuant l'estat neutre) que ofereix la base de dades RAVDESS: enuig, alegria, tristesa, calma, por, indignació i sorpresa. Els paràmetres escollits són 22.050 Hz de mostreig, 23 ms de temps de finestra i 100 Hz de tall del filtre. En aquest cas el total de dades ascendeix a 129024.

La precisió total és de 73.51 % amb 165 paràmetres. En aquest cas hi ha moltes classes que el sistema SVM ha de comparar una contra la resta. Probablement, el resultat podria millorar canviant aspectes del classificador com el grau del kernel polinòmic o, inclòs provar amb altres classificadors. També es poden incorporar noves característiques, que diferencien millor cada emoció i nous mètodes de selecció de característiques i paràmetres estadístics. La matriu de confusió aporta una visió del comportament del classificador amb 7 emocions:

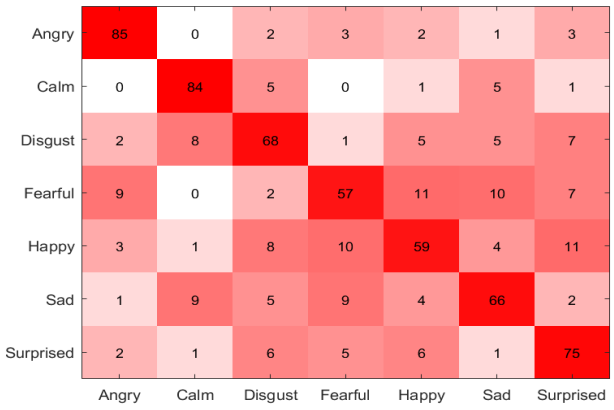

Figura 26: Matriu de confusió amb 7 emocions

- Es fa evident que el sistema diferencia millor entre emocions amb nivell d'excitació diferents. Per exemple, no hi ha confusió entre enuig i calma o, por i calma

Capítol 9 Conclusions

En aquest capítol es valora la feina realitzada i les conclusions que se'n deriven. En l'apartat 9.1, s'analitza l'èxit del projecte en quant al acompliment dels objectius, la planificació i la metodologia. En el 9.2, es presenten les conclusions del treball. Finalment, en la secció 9.3, s'enumeren algunes de les línies de treball futur tant a nivell pràctic com teòric.

9.1 Acompliment dels objectius

Aquest projecte parteix de dos objectius principals i un de secundari. El primer objectiu és obtenir un programari que permeti reconèixer l'emoció subjacent en una frase d'àudio. Aquest s'ha complert de forma satisfactòria juntament amb l'objectiu secundari: implementar una interfície gràfica. Per aconseguir-ho s'ha estudiat l'eina de programació utilitzada, el MATLAB. Prèviament, s'ha assolit una base teòrica en l'àmbit del processament del senyal d'àudio, l'extracció i selecció de característiques acústiques i sistemes de classificació. Amb aquest estudi i posterior implementació s'ha superat el segon objectiu.

Per a l'acompliment dels objectius s'ha seguit la metodologia i planificació esmentades en la introducció d'aquest document. Cal afegir que per a aconseguir l'objectiu secundari, implementació de la GUI (amb el codi 3.4), programat durant l'etapa d'implementació (veure figura 1: Diagrama de Gantt del TFG), s'ha hagut de desplaçar la feina a l'etapa de redacció de la memòria. Això ha estat degut a una incidència amb els recursos materials que ha provocat l'endarreriment en la presentació i documentació de la GUI i un increment del temps de dedicació diari durant la redacció de la memòria.

9.2 Conclusions

En aquest sistema s'analitzen 16 característiques acústiques: el *pitch* més la seva derivada, l'energia més la seva derivada i 12 coeficients LFPC. I, s'extreuen 12 paràmetres estadístics de cada característica; que fan un total de 192 paràmetres. Per reduir el nombre de característiques, aquests paràmetres s'ordenen per rellevància segons el mètode d'informació mútua. Finalment, s'entrenen els classificadors SVM afegint les característiques de manera seqüencial per a cada entrenament i s'escull com a model aquell amb el valor de precisió més alt.

El sistema permet l'obtenció de models per a diferents bases de dades de veus emocionades i per a diferents conjunts d'emocions, a més d'incloure l'opció d'introduir paràmetres opcionals per al preprocessament. Per a la realització d'aquest document (a excepció de la prova 3) totes les figues i taules mostrades en la implementació fan referència a:

- 4 emocions: enuig, alegria, tristesa i neutre
- 96 observacions o frases per cada emoció
- Freqüència de mostreig : 22050 Hz
- Temps de finestra: 23 ms
- Freqüència de tall del filtre: 100 Hz

Segons això, s'han representat les distribucions de densitat de cada característica en les figures 17, 18, 19, 20 i 22 i es mostren els paràmetres més rellevants per a les prediccions en la taula 4. Analitzant la taula 4, les distribucions, i els experiments, es poden extreure diverses conclusions:

- Donat que amb les 15 primeres característiques els models ja obtenen fins a un 75% de precisió, es dedueix que els paràmetres amb més informació del contingut emocional en la parla (per al conjunt de 4 emocions) són els coeficients LFPC2 i LFPC1, que apareixen 5 i 4 vegades, respectivament, entre les 15 primeres característiques de la taula 4.

- Aquests coeficients representen la distribució energètica dins el marge freqüencial de 100 a 230 Hz. Si es contrasta aquesta informació amb la prova 2 es ratifica aquest fet; ja que filtrar les freqüències per sota dels 200 Hz significa perdre fins a un 3% de precisió.

- El paràmetre que té menys pes en la taula 4 és F0_MAX, el valor de *pitch* màxim. Observant els resultats, aquest valor coincideix per a totes les observacions de totes les classes (393,75 Hz). Això, està provocat pel mètode amb el que s'extreu el *pitch*. De fet aquesta freqüència coincideix amb la freqüència màxima dins el vector de temps que s'utilitza per a calcular el *pitch.* Així doncs, per a aquells segments on la freqüència amb valor de pic més gran sigui igual o més gran que 393,75 Hz, el resultat serà aquest valor. Per tant, aquest paràmetre es descartable.

- Totes les proves han demostrat que es classifiquen millor les emocions amb diferents nivells d'excitació. Això, pot ser conseqüència de que la majoria de característiques acústiques analitzades en aquest projecte fan referència a l'energia del senyal (ja sigui de tot un segment o d'un domini freqüencial concret del segment).

- S'ha demostrat que amb temps de finestra petits s'aconsegueix més precisió. És a dir: amb molts fragments amb poques mostres cada un. Es dedueix que la informació temporal és important i que no tota la informació sobre les emocions es troba en freqüències baixes.

- Amb freqüències de mostreig de 16000 Hz la precisió es redueix en un 3%. Per una banda, es perd estructura degut a la menor quantitat de mostres i per l'altra, es perd tota la informació per sobre dels 8000 Hz. Això porta a la conclusió de que a freqüències altes també hi ha informació valuosa sobre l'estat emocional, cosa que confirma el punt anterior.

- Es pot assegurar, per tant, que les emocions afecten a diferents rangs freqüencials de la veu i s'han de tenir tots en compte.

- L'energia és una característica clau per a la diferenciació entre emocions.

9.3 Línies de treball futur

Les possibilitats per a la continuïtat del projecte són moltes. A nivell de recerca es pot aprofundir en diferents disciplines i conceptes d'aquest projecte, com:

- les emocions i la relació amb la fisiologia de l'aparell fonador.
- l'acústica fonètica, per tal de trobar aquelles característiques que més informació aporten en referència a les emocions.
- mètodes de selecció de característiques que permetin obtenir el rendiment màxim amb el mínim d'elements possibles.
- Sistemes de classificació.

A un nivell més pràctic, el sistema proposat permet l'ampliació del sistema gràcies a la modularitat del disseny i l'abstracció entre els diferents processos. A continuació s'enumeren diverses possibilitats:

- Introduir més paràmetres opcionals per variar el comportament d'altres submòduls. Per exemple: escollir entre les característiques a extreure i els paràmetres estadístics d'aquestes o determinar el funcionament del mètode de selecció de característiques i del sistema de classificació. Això permetria incrementar el nombre d'experiments i optimitzar el sistema.
- Ampliar la quantitat de característiques acústiques. Per exemple amb els coeficients MFCC (*Mel Frequency Cepstrum Coefficients*), una característica molt utilitzada en el reconeixement de la veu en general, o els tres primers formants de la veu o harmònics de la freqüència fonamental.
- Implementar més mètodes de selecció de característiques i permetre escollir entre aquests. Alguns dels emprats en treballs similars, que és basen en algoritmes genètics, redueixen el conjunt de característiques en un 89%.
- Utilitzar diferents sistemes de classificació. Una opció molt valorable, que s'ha vist en alguns treballs d'aquesta bibliografia, és l'ús de més d'un sistema de classificació. Per exemple: un sistema previ a l'implementat en aquest projecte,

que classifiqui les emocions segons l'excitació i les característiques acústiques directament; el sistema HMM aporta bons resultats en alguns dels treballs estudiats.

- Provar el sistema amb diferents bases de dades. Una de les més utilitzades en estudis similars és la *Berlin DataBase of Emotional Speech.*

Bibliografia

[1] R. Cowie, et al., 2001. Emotion recognition in human–computer interaction, IEEE Signal Processing Magazine 18, 32–81.

[2] B. Yang, M. Lugger, 2010. Emotion recognition from speech signals using new harmony features, Signal Processing 90, 1415–1423.

[3] Ververidis, D., Kotropoulos, C., 2006. Emotional speech recognition: resources, features, and methods. Speech Communication 48, 1162–1181.

[4] Eckman, P., 1992. An argument for basic emotions. Cognition and Emotion 6 (3/4), 169-200.

[5] Cornelius, R., 1996. The Science of Emotion. Research and Tradition in the Psychology of Emotion. Prentice-Hall, Upper Saddle River, NJ.

[6] Cowie, R., Cornelius, R., 2003. Describing the emotional states that are expressed in speech. Speech Communication 40, 5–32.

[7] Alexander I. Iliev, et al., 2010. Spoken emotion recognition through optimum-path forest classification using glottal features. Computer Speech and Language 24, 445–460.

[8] Noda, T., Yano, Y., Doki, S., Okuma, S., 2006. Adaptive emotion recognition in speech by feature selection based on KL-divergence. In: IEEE International Conference on Systems, Man, and Cybernetics, 1921-1926, October 8–11, Taipei, Taiwan.

[9] Cobeta, I., Nuñez, F., Fernández, S., (coord.) 2013. Patología de la Voz (1ª edició). Editorial: Marge Médica Books (Barcelona).

[10] Moataz El Ayadi, et al., 2011. Survey on speech emotion recognition: Features, classification schemes, and databases. Pattern Recognition 44, 572–587.

[11] L. Rabiner, R. Schafer, Digital Processing of Speech Signals, first ed., Pearson Education, 1978.

[12] M.T. Shami, M.S. Kamel, Segment-based approach to the recognition of emotions in speech, in: IEEE International Conference on Multimedia and Expo, 2005. ICME 2005, 2005, 4pp.

[13] T. Nwe, S. Foo, L. De Silva, Speech emotion recognition using hidden Markov models, Speech Commun. 41 (2003) 603–623.

[14] Planet Garcia, S., Morán Moreno, Jose A., Formiga Fanals, Ll. (2003) *Reconocimiento de emociones basado en el análisis de la señal de voz parametrizada.* Enginyeria i Arquitectura La Salle – Universitat Ramon Llull (PGR-BP-2003-03).

[15] M. Lugger, B. Yang, Combining classifiers with diverse feature sets for robust speaker independent emotion recognition, in: Proceedings of EUSIPCO, 2009.

[16] Sidorova, J., Optimization Techniques for Speech Emotion Recognition. *Departament de Traducció i Ciències del Llenguatge.* TESI DOCTORAL UPF / 2009 Director de la tesis: Prof. PhD Toni Badia.

[17] Livingstone, S. R., Peck, K., & Russo, F. A. (2012). *RAVDESS: The Ryerson Audio-Visual Database of Emotional Speech and Song*. Paper presented at the 22nd Annual Meeting of the Canadian Society for Brain, Behaviour and Cognitive Science (CSBBCS), Kingston. [Disponible *online*: http://smartlaboratory.org/ravdess/home/]

Annex 1 Codi font

El projecte s'ha dut a terme amb la versió R2016a de MATLAB, fent servir les *toolbox* d'ús gratuït per a estudiants de la UOC. Totes les funcions que s'adjunten han estat creades de nou per al projecte a excepció de **endPointDetect.m** (Roger Jang, 2004 -amb modificacions per a aquest projecte-), **heatmap.m** (The MathWorks, Inc. 2009-2014), **MI.m** (Okko Rasanen, 2013) i **trainClassifier.m** (generada amb l'aplicació *Classification Learner* -amb modificacions per a aquest projecte-).

```
function [model,accuracy] = SER_TRAINING( main_directory,fs,frame_size,HPF_fc,fid,CTSout)
% [model,accuracy] = SER_TRAINING( main_directory,fs,frame_size,HPF_fc)
% Exectute the training phase
%
%   Inputs:
%           main_directory:     directory with subdirectorys (each subdir
%                               must be a class with the same num of fiels)
%           fs:                 sample frecuency
%           frame_size:         time in s that determines the window size
%           HPF_fc:             cutoff frecuency of hight pass filter
%           fid:                ID log file used in CTSSER GUI
%           CTSout:             display out of CTSSER GUI
%
%   Outputs:
%           model:              model to make predictions with
%                               SER_PREDICT.m
%           accuracy:           accuracy of the model
%
%
% Author: Marcel Gonzalez Poy, 2016.

if nargin<2, fs=22050; end
if nargin<3, frame_size=0.023; end
if nargin<4, HPF_fc=100; end
if nargin<5, fid=1; end
if nargin<6, CTSout=''; end


fprintf(fid,'Loading data\n');
if fid~=1
    CTS_listboxDraw(fid,CTSout);
end
repository = LOADDATA( main_directory );
labels=fieldnames(repository);


fprintf(fid,'\nExtracting and selecting features\n');
if fid~=1
    CTS_listboxDraw(fid,CTSout);
end
[table,features,~]=EXTRACTION_MODULE(repository,fs,frame_size,HPF_fc,fid,CTSout);

fprintf(fid,'\nTraining system\n');
if fid~=1
    CTS_listboxDraw(fid,CTSout);
end
[model,validationPredictions,accuracy] = CLASSIFICATION_MODULE(table,features,labels,fid,CTSout);

% create field opts and accuracy
model.opts.fs=fs;
model.opts.frame_size=frame_size;
model.opts.HPF_fc=HPF_fc;
model.Accu=accuracy;

%plot confusion matrix
figure('Name','Confusion Matrix');
confMat=confusionmat(model.ClassificationSVM.Y,validationPredictions);
heatmap(confMat, model.ClassificationSVM.ClassNames, model.ClassificationSVM.ClassNames, 1,...
    'Colormap','red','ColorLevels',128,'ShowAllTicks',1,'UseLogColorMap',true);

end
```

```
function prediction = SER_PREDICT(filename,model)
% prediction = SER_PREDICT(filename,model)
% Exectute the prediction phase
%
%   Inputs:
%           filename:       audio file
%           model:          model for make prediction
%
%   Outputs:
%           prediction:     prediction
%
%
% Author: Marcel Gonzalez Poy, 2016.

unknownData = LOADFILE(filename);
fs=model.opts.fs;
HPF_fc=model.opts.HPF_fc;
frame_size=model.opts.frame_size;
```

```
[table,~,~]=EXTRACTION_MODULE(unknownData,fs,frame_size,HPF_fc);

prediction=model.predictFcn(table);

end
```

```
function repository = LOADDATA( main_directory )
%repository = LOADDATA( main_directory )
% load all data files form directory to the project
%
%   Inputs:
%           main_directory:     directory with subdirectorys (each subdir
%                               must be a class with the same num of fiels)
%
%   Outputs:
%           repository:         struct of audio files for each class
%
%
% Author: Marcel Gonzalez Poy, 2016.

getDir=dir(main_directory);
for i=1:length(getDir);
    if(getDir(i).isdir==1)
        dir_name=getDir(i).name;
        if(strcmp('..',dir_name)==0 && strcmp('.',dir_name)==0)
            AudioData=strcat(main_directory,'\',dir_name,'\*.wav');
            repository.(dir_name)=dir(AudioData);
        else
            continue;
        end
    end

end
end
```

```
function unknownData = LOADFILE(filename)
%unknownData = LOADFILE(filename)
% load an audio file
%
%   Inputs:
%           filename:      filename of an audio
%
%   Outputs:
%           unknownData:   struct with audio file
%
%
% Author: Marcel Gonzalez Poy, 2016.
 unknownData.file=filename;

end
```

```
function[table,features,features_extracted]=EXTRACTION_MODULE(repository,fs,frame_size,HPF_fc,fid,CTSout)
%[table,features]=EXTRACTION_MODULE(repository,fs,HPF_fc,frame_size)
%Execute the extraction module of SER system
%
%   Inputs:
%           repository:     struct of speech data created by LOADDATA.m
%           fs:             sample frecuency
%           HPF_fc:         cutoff frecuency of hight pass filter
%           frame_size:     time in ms that determines the window size
%           fid:                ID log file used in CTSSER GUI
%           CTSout:             displau out of CTSSER GUI
%
%   Outputs:
%           table:              table (columns: statistics, rows: observations).
%                               Contains name label and code label
%           features:           vector of features in order of relevance (based
%                               in mutual information)
%           features_extracted: struct of features extracted for each class
%
% Author: Marcel Gonzalez Poy, 2016.


labels=fieldnames(repository);          %get the labels (classes)
repo_cell=struct2cell(repository);

frame_shift=frame_size/2;               %overlaping time = 1/2 window time

% Process each class
for i=1:length(repo_cell)
```

```
    %Display the progres in command window for each class.
    if(length(repo_cell)>1)
        data=struct2cell(repo_cell{i});
        fprintf(fid,'\tProcessing files form %s directory\n ',labels{i});
        if fid~=1
            CTS_listboxDraw(fid,CTSout);
        end

    else
        data=repo_cell;
    end
    %Process each audio of current class
    for j=1:length(data)

     winFrameMatrix =PREPROCESS(data{1,j},fs,frame_size,frame_shift,HPF_fc);
    [features_extracted.(labels{i})(j),statistics.(labels{i})(j)]=FEATURES_EXTRACTION(winFrameMatrix,fs);

    end

end

% generate the table
table=stats_table(statistics);

if(length(repo_cell)>1)
    % generate the features vector
    features= FEATURES_SELECTION(table);
else
    features=[];
end

end
```

```
function[SVMmodel,validationPredictions,accuracy_max] =
CLASSIFICATION_MODULE(table,features,labels,fid,CTSout)
%[SVMmodel,validationPredictions,accuracy_max] = CLASSIFICATION_MODULE(table,features,labels)
% Execute the classification module
%
%   Inputs:
%           table:          table of observations x statistics
%           features:       vector of features selected
%           labels:         class names
%           fid:                ID log file used in CTSSER GUI
%           CTSout:             displau out of CTSSER GUI
%
%   Outputs:
%           SVMmodel:                   best SVM model
%           validationPredictions:      vector of predictions
%           accuracy_max:               accuracy value of best SVMmodel
%
% Author: Marcel Gonzalez Poy, 2016.

addFeature=[];
accuracy_max=0;

for i=1:length(features);

    addFeature=[addFeature;features(i)];
    fSel_pos=[2;addFeature+2];
    reducedTable=table(1:end,fSel_pos);

    featuresSelected=table.Properties.VariableNames(fSel_pos(2:end));

[SVMmodel_temp,accuracy_temp,validationPredictions_temp]=trainClassifier(reducedTable,featuresSelected,..
labels);

    if (accuracy_temp>accuracy_max)
        SVMmodel=SVMmodel_temp;
        accuracy_max=accuracy_temp;
        validationPredictions=validationPredictions_temp;
        fprintf(fid,'\n\t%.2f %% accuracy with %d features\n',accuracy_max*100,i);
        if fid~=1
            CTS_listboxDraw(fid,CTSout);
        end
    end
end

end
```

```
function winFramesMatrix= PREPROCESS(filename,fs,frame_size,frame_shift,HPF_fc)
%winFrameMatrix= PREPROCESS(filename,fs,HPF_fc,frame_size,frame_shift)
```

```
%Execute the preprocess submodule of the extraction module
%
%   Inputs:
%           filename:       audio file fs:              sample frecuency
%           HPF_fc:         cutoff frecuency of hight pass filter
%           frame_size:     time in s that determines the window size
%           frame_shift:    time in s that determines the window
%           overlapping
%   Outputs:
%           winFrameMatrix: matrix of frames X samples windowed with
%           Hamming
%
% Author: Marcel Gonzalez Poy, 2016.

[data,original_fs]=audioread(filename);

% ===== HPF
filtered_data=hpf(data,original_fs,HPF_fc);

% ===== Normalize
normailzed_data=filtered_data/max(abs(filtered_data));

% ===== Resampling
if (original_fs>fs)
 [p,q]=rat(fs/original_fs,0.0001);
 resampled_data=resample(normailzed_data,p,q);
else
 resampled_data= normailzed_data;
 fs=original_fs;
end

% ===== End Point detection
[endPoints,~]=endPointDetect(resampled_data,fs);
eP_data=resampled_data(endPoints(1):endPoints(2));

% ===== Segmentation and windowing

winFramesMatrix= windowing (eP_data,frame_size,frame_shift,fs);

end
```

```
function [features_extracted,statistics] = FEATURES_EXTRACTION(frame_matrix,fs)
%[features_extracted,statistics] = FEATURES_EXTRACTION(frame_matrix,fs)
% Extract features from a framed audio
%
%   Inputs:
%           frame_matrix:   matrix frames x samples fs:             sample
%           rate
%   Outputs:
%           features_extracted:     struct of features of an audio
%           statistics:             struct of statistics of an audio
%
% Author: Marcel Gonzalez Poy, 2016.

% ==== Pitch
features_extracted.F0=pitch(frame_matrix,fs);
features_extracted.dF0=diff(features_extracted.F0);

% ==== log energy
features_extracted.logE=log_energy(frame_matrix);
features_extracted.dlogE=diff(features_extracted.logE);

% ==== short term log frequency power coefficients
LFPC=lfpc(frame_matrix,fs)';
[m,n]=size(LFPC);
for i=1:m
    features_extracted.(strcat('LFPC',int2str(i)))=LFPC(i,1:n);
end

% ==== statistic compute

statistics=calc_statistics(features_extracted);

end
```

```
function [features,w] = FEATURES_SELECTION( table )
%[features,w] = FEATURES_SELECTION( table )
% Sort features of all observations by its relevance
%
%   Inputs:
%           table:          table of observations x statistics
%   Outputs:
```

```
%           features:       F x 1 sized vector of feature indices in the
%                              descending order of relevance.
%           w:              F x 1 sized vector of feature relevances (MIs)
%           in the
%                              descending order.
%
% Author: Marcel Gonzalez Poy, 2016.

labels=table2array(table(1:end,1));

data=table2array(table(1:end,3:end));

[features,w]=MI(data,labels);

end
```

```
function HPF_data = hpf( data, fs, fc )
% HPF_data = hpf( data, fs, fc )
%Implement a elliptic high pass filter
%
%   Inputs:
%           data:       audio data fs:         sample rate fc:
%           cutoff frecuency of the filter
%   Outputs:
%           HPF_data:   filtered audio
%
% Author: Marcel Gonzalez Poy, 2016.

FCN=((2*pi*fc)/fs)/pi;
[z,p]=ellip(6,0.1,60,FCN,'high');
HPF_data = filter(z,p,data);

end
```

```
function [endPoint, voicedSegment] = endPointDetect(y, fs, nbits, plotOpt, epdParam)
% endPointDetect: End point detection based on volume and zero-crossing
% rate
%   Usage: [endPoint, voicedSegment] = endPointDetect(y, fs, nbits,
%   plotOpt, epdParam)

%   Roger Jang, 2004/11/18
%   lines: 101, 112, 126 modified by Marcel Gonzalez Poy, 2016

if nargin<1, selfdemo; return; end
if nargin<2, fs=22050; end
if nargin<3, nbits=16; end
if nargin<4, plotOpt=0; end
if nargin<5,
    epdParam.frameSize = round(fs/31.25);   % fs=8000 ===> frameSize=256;
    epdParam.frameSize = 320;       % same as htk
    epdParam.overlap = round(epdParam.frameSize*3/4);
    epdParam.zcrRatio = 0.3;
    epdParam.testFrameNum = 5;      % ´ú¸Õ¥§¡Àô¹Ò¾¸µª°µ©Ø¼Æ¡]¨S¥Î¨ì¡^
    epdParam.volRatio1=2;
    epdParam.volRatio2=4;
end

frameSize=epdParam.frameSize;
overlap=epdParam.overlap;
zcrRatio=epdParam.zcrRatio;

% ====== Zero adjusted
y = y-round(mean(y));

% ====== Take frames
framedY  = buffer(y, frameSize, overlap);
frameNum = size(framedY, 2);            % Number of frames

% ====== Compute volume and zcr
volume=zeros(frameNum, 1);
zcr=zeros(frameNum, 1);
for i=1:frameNum
    frame=framedY(:, i);
    frame=frame-round(mean(frame));
    volume(i)=sum(abs(frame-mean(frame)))/frameNum;
    zcr(i)=sum(abs(diff(frame>0)));
end
frameTime = frame2sampleIndex(1:frameNum, frameSize, overlap)/fs;
volTh=max(volume);
volTh1=epdParam.volRatio1*volTh;
volTh2=epdParam.volRatio2*volTh;
```

```
% ====== Find initial end points according volume level2 (upper level)
voicedIndex = find(volume>=volTh2);
while isempty(voicedIndex)  % Y§ä¤£¨ì¦³Ánµ³¡¤À¡A§ïÅÜµ¶q¤ùÂeÈ
    volTh1=max(volume)/16;
    volTh2=max(volume)/8;
    voicedIndex = find(volume>=volTh2);
end

zcrTh = max(zcr)*zcrRatio;
sound = [];
k = 1;
sound(k).begin = voicedIndex(1);
for i=2:length(voicedIndex)-1,
    if voicedIndex(i+1)-voicedIndex(i)>1,
        sound(k).end = voicedIndex(i);
        sound(k+1).begin = voicedIndex(i+1);
        k = k+1;
    end
end
sound(k).end = voicedIndex(end);

% ====== Delete short sound clips
%index = []; for i=1:length(sound),
%   if (sound(i).end-sound(i).begin)<4
%       index = [index, i];
%   end
%end sound(index) = [];

% ====== Expand end points to volume level1 (lower level)
for i=1:length(sound),
    head = sound(i).begin;
    while (head-1)>=1 && volume(head-1)>volTh1,
        head=head-1;
    end
    sound(i).begin = head;
    tail = sound(i).end;

    while (tail+1)<=length(volume) && volume(tail+1)>volTh1,
        tail=tail+1;
    end
    sound(i).end = tail;
end

% ====== Expand end points to include high zcr region
for i=1:length(sound),
    head = sound(i).begin;
    while (head-1)>=1 && zcr(head-1)>zcrTh,
        head=head-1;
    end
    sound(i).begin = head;
end

% ====== Delete repeated sound segments
if ~isempty(sound) %modified by Marcel Gonzalez Poy, 2016. original: if length(sound)~=0
    index = [];
    for i=1:length(sound)-1
        if sound(i).begin==sound(i+1).begin && sound(i).end==sound(i+1).end,
            index=[index, i];
        end
    end
    sound(index) = [];
end;

% ====== Transform sample-point-based index
if ~isempty(sound) %modified by Marcel Gonzalez Poy, 2016. original: if length(sound)~=0
    for i=1:length(sound)
        voicedSegment(i).begin = sound(i).begin*(frameSize-overlap);
        voicedSegment(i).end   = min(length(y), sound(i).end*(frameSize-overlap));
    end
    endPoint=[voicedSegment(1).begin, voicedSegment(end).end];  % ¨úÀY§À
else
    endPoint = [];
end;

if plotOpt,
    axes1H=subplot(4,1,1);
    time=(1:length(y))/fs;
    plot(time, y); grid on
    axis([min(time), max(time), min(y), max(y)]); %modified by Marcel Gonzalez Poy, 2016.
      %original: axis([min(time), max(time), -2^nbits/2, 2^nbits/2])
    ylabel('Amplitude');
    title('Waveform');
    % Plot end points
    yBound=[-1, 1];
    for i=1:length(sound),
```

```
        line(frame2sampleIndex(sound(i).begin, frameSize, overlap)/fs*[1,1], yBound, 'color', 'm');
        line(frame2sampleIndex(  sound(i).end, frameSize, overlap)/fs*[1,1], yBound, 'color', 'g');
    end

    axes2H=subplot(4,1,2);
    plot(frameTime, volume, '.-'); grid on
    line([min(frameTime), max(frameTime)], volTh1*[1 1], 'color', 'c');
    line([min(frameTime), max(frameTime)], volTh2*[1 1], 'color', 'c');
    axis tight
    ylabel('Volume');
    title('Volume');
    % Plot end points
    yBound = [min(volume) max(volume)];
    for i=1:length(sound),
        line(frame2sampleIndex(sound(i).begin, frameSize, overlap)/fs*[1,1], yBound, 'color', 'm');
        line(frame2sampleIndex(  sound(i).end, frameSize, overlap)/fs*[1,1], yBound, 'color', 'g');
    end

    axes3H=subplot(4,1,3);
    plot(frameTime, zcr, '.-'); grid on
    line([min(frameTime), max(frameTime)], zcrTh*[1 1], 'color', 'c');
    axis([min(frameTime), max(frameTime), 0, max(zcr)]);
    ylabel('ZCR');
    title('Zero crossing rate');
    % Plot end points
    yBound = [0 max(zcr)];
    for i=1:length(sound),
        line(frame2sampleIndex(sound(i).begin, frameSize, overlap)/fs*[1,1], yBound, 'color', 'm');
        line(frame2sampleIndex(  sound(i).end, frameSize, overlap)/fs*[1,1], yBound, 'color', 'g');
    end

    axes4H=subplot(4,1,4);
    voicedIndex=endPoint(1):endPoint(2);
    voicedTime=time(voicedIndex);
    voicedY=y(voicedIndex);
    voicedH=plot(voicedTime, voicedY); grid on
    axis([time(endPoint(1)), time(endPoint(2)), -inf, inf]);
    ylabel('Amplitude');
    title('Voiced waveform');

    U.y=y; U.fs=fs; U.nbits=nbits;
    U.axes1H=axes1H; U.axes2H=axes2H; U.axes3H=axes3H; U.axes4H=axes4H;
    U.voicedIndex=voicedIndex; U.voicedH=voicedH;
    U.voicedY=voicedY; U.voicedTime=voicedTime;
    set(gcf, 'userData', U);

    %uicontrol('string', 'Play all', 'callback', 'U=get(gcf, ''userData'');
    %sound(U.y/(2^U.nbits/2), U.fs);'); uicontrol('string', 'Play voiced',
    %'callback', 'U=get(gcf, ''userData''); sound(U.voicedY/(2^U.nbits/2),
    %U.fs);', 'position', [100, 20, 60, 20]);

    % Play the segmented sound
%   head = sound(1).begin*(frameSize-overlap); tail = min(length(y),
%   sound(end).end*(frameSize-overlap)); thisY = y(head:tail); fprintf('His
%   return to hear the cutted sound %g:', i); pause; fprintf('\n');
%   wavplay(thisY, fs, 'sync'); fprintf('\n');
end

% ======
function sampleIndex=frame2sampleIndex(frameIndex, frameSize, overlap)
sampleIndex=(frameIndex-1)*(frameSize-overlap)+round(frameSize/2);

% ====== Self demo function selfdemo waveFile='malisa\SenPC0000_2.wav';
% waveFile='Arthas\SenIC0000_3.wav'; waveFile='tracy\SenIC0000_2.wav';
% waveFile='abo\SenIC0000_3.wav'; waveFile='roger\SenIC0000_2.wav';
% waveFile='jacky\SenIC0000_3.wav'; waveFile='__Cyberon__\2_f_0.wav';
% waveFile='__Cyberon__\1_f_19.wav'; waveFile='¥D¤H¤U°¨«Ë¦b²î.wav';
% waveFile='¦¹«ë°ø°øµLµ´´Ä+sil.wav'; waveFile='·¨®a¦³¤k¤ì¤ø¦¨.wav'; plotOpt
% = 1; [y, fs, nbits] = wavReadInt(waveFile); endPoint = feval(mfilename,
% y, fs, nbits, plotOpt);

end
end
```

```
function WinFramesMatrix= windowing (data,frame_size,frame_shift,fs)
%WinFramesMatrix= windowing (data,frame_size,frame_shift,fs)
%
% Segemntation and  Hamming windowing of an audio
%
%   Inputs:
%           data:               audio data fs:                  sample
%           frecuency frame_size:         time in s that determines the
%           window size frame_shift:        time in s that determines the
%           window overlapping
```

```
%   Outputs:
%          WinFrameMatrix:      matrix frames x samples of processed audio
%
%
% Author: Marcel Gonzalez Poy, 2016.

N_samples_window = fix(frame_size*fs);            %Number of Samples/window
N_overlap_samples = fix(frame_shift*fs);          %Number of overlaping samples
N_segments = fix((length(data)-N_samples_window)/N_overlap_samples)+1;%Number of segments in speechsignal
WIN = hamming(N_samples_window);    %hamming window of N_samples_window points

% ==== Create a Matrix of windowed frames (rows: frames, columns: samples)
WinFramesMatrix= zeros(N_segments,N_samples_window);

for i = 1:N_segments
    Index = (i-1)*N_overlap_samples+1;
    WinFramesMatrix(i,1:N_samples_window)= data(Index:Index+N_samples_window-1).*WIN;

end

end
```

```
function F0=pitch(frame_matrix,fs)
%F0=pitch(frame_matrix,fs) Estimate fundamental frequancy based on complex
%cepstrum
%
%   Inputs:
%           frame_matrix:   matrix of frames x samples fs:
%           sample frecuency
%   Outputs:
%           F0:             vector of F0 of each frame
%
% Author: Marcel Gonzalez Poy, 2016.
%Reference:
%http://es.mathworks.com/help/signal/ug/estimating-fundamental-frequency-with-the-complex-
% cepstrum.html?searchHighlight=cepstrum

[m,n]=size(frame_matrix);
dt=1/fs;
F0=zeros(1,m);

for i=1:m

    frame=frame_matrix(i,1:n);
    c=cceps(frame);

    t = 0:dt:length(frame)*dt-dt;

    trng = t(t>=2.5e-3 & t<=16e-3);
    crng = c(t>=2.5e-3 & t<=16e-3);

    [~,I] = max(crng);

    F0(i)=1/trng(I);

end
end
```

```
function logE= log_energy(frame_matrix)
%logE= log_energy(frame_matrix)
% Ectract log energy feature of a processed audio
%
%   Inputs:
%           frame_matrix:   matrix frames x samples
%
%   Outputs:
%           logE:           feature vector
%
%
% Author: Marcel Gonzalez Poy, 2016.

[m,n]=size(frame_matrix);
logE=zeros(1,m);

for i=1:m

    frame=frame_matrix(i,1:n);
    logE(i)=log(sum(frame.^2));

end

end
```

```
function [Filter,elemF,N] = lfpc_bankFilter(fVector,init_bw,init_cf,alfa)
%[Filter,elemF,N] = lfpc_bankFilter(fVector,init_bw,init_cf,alfa)
% Ompliement a bank of filters for LFPC extraction
%
%   Inputs:
%           fVector:        frecuency vector init_bw:        bandwith of
%           the first filter init_cf:        cuttoff frecuency of the first
%           filter alfa:           logarithmic growth factor
%   Outputs:
%           Filter:         bank of filters elemF:          numnber of
%           frecuency elements of each filter
%                           for given fVector
%           N:              number of filtres of the bank
%
% Author: Marcel Gonzalez Poy, 2016.
%
% Refecrence: [1] T. Nwe, S. Foo, L. De Silva, Speech emotion recognition
% using hidden Markov models,
%                 Speech Commun. 41 (2003) 603-623.

if nargin<2, init_bw=54; end
if nargin<3, init_cf=127; end
if nargin<4, alfa=1.4; end


N=12;

cf=zeros(1,N);
bw=zeros(1,N);
lf=zeros(1,N);
hf=zeros(1,N);
elemF=zeros(1,N);
Filter=zeros([N numel(fVector)]);

cf(1)=init_cf;
bw(1)=init_bw;
lf(1)=cf(1)-(bw(1)/2);
hf(1)=cf(1)+(bw(1)/2);

ifilter=find(fVector>=lf(1) & fVector<=hf(1));
elemF(1)=numel(ifilter);
Filter(1,ifilter)=rectwin(elemF(1));

for i=2:N

    bw(i)=alfa*bw(i-1);
    sum=0;
    for j=1:(i-1)
        sum=sum+bw(j);
    end
    cf(i)=cf(1)+((bw(i)-bw(1))/2)+sum;
    lf(i)=cf(i)-(bw(i)/2);
    hf(i)=cf(i)+(bw(i)/2);

    ifilter=find(fVector>=lf(i) & fVector<=hf(i));
    elemF(i)=numel(ifilter);
    Filter(i,ifilter)=rectwin(elemF(i));

end
end
```

```
function LFPC = lfpc(frame_matrix,fs)
%LFPC = lfpc(frame_matrix,fs) Extrat coefficients LFPC from processed audio
%
%   Inputs:
%           frame_matrix:   matrix frames x samples fs:             sample
%           rate
%   Outputs:
%           LFPC:           matrix of LFPC coefficients
%
% Author: Marcel Gonzalez Poy, 2016. Refecrence: [1] T. Nwe, S. Foo, L. De
% Silva, Speech emotion recognition using hidden Markov models,
%                 Speech Commun. 41 (2003) 603-623.

[m,n]=size(frame_matrix);
NFFT=2^nextpow2(n);
fVector=fs*(0:(NFFT/2))/NFFT;
[Filter,elemF,N]=lfpc_bankFilter(fVector);
LFPC=zeros([m N]);

for i=1:m

    frame=frame_matrix(i,1:n);
```

```
    Yframe=fft(frame,NFFT);
    P_Yframe=abs(Yframe/NFFT);



    for j=1:N;
        Sf=Filter(j,:).*P_Yframe(1:(NFFT/2+1));
        P_Sf=sum(Sf.^2);
        LFPC(i,j)=db(P_Sf,'power')/elemF(j);
    end

end

end
```

```
function statistics = calc_statistics( features_extracted )
%function statistics = calc_statistics( features_extracted ) Compute the
%statistics of all features extracted from audio file
%
%   Inputs:
%           features_extracted: struct of features
%
%   Outputs:
%           statistics:         struct of statsistics
%
% Author: Marcel Gonzalez Poy, 2016.

f_names=fieldnames(features_extracted);

for i=1:numel(f_names)

    data=features_extracted.(f_names{i});
    [m,n]=size(data);

    for j=1:m
        statistics.(f_names{i})(j)=aux_stats(data(j,1:n),f_names{i});
    end

end

end
```

```
function stats = aux_stats( vector, featureName )
%function stats = aux_stats( vector, featureName ) Compute the statistics
%of a vector
%
%   Inputs:
%           vector:         feature vector featureName:    feature vector
%           name
%
%   Outputs:
%           stats:         struct of statsistics
%
% Author: Marcel Gonzalez Poy, 2016.

stats.(strcat(featureName,'_MEAN'))=mean(vector);
stats.(strcat(featureName,'_MIN'))=min(vector);
stats.(strcat(featureName,'_MAX'))=max(vector);
stats.(strcat(featureName,'_STD'))=std(vector);
stats.(strcat(featureName,'_SKEW'))=skewness(vector);
stats.(strcat(featureName,'_KURT'))=kurtosis(vector);
quart=quantile(vector,3);
    stats.(strcat(featureName,'_Q1'))=quart(1);
    stats.(strcat(featureName,'_Q2'))=quart(2);
    stats.(strcat(featureName,'_Q3'))=quart(3);
stats.(strcat(featureName,'_IQR'))=iqr(vector);
stats.(strcat(featureName,'_SID'))=iqr(vector)/2;
stats.(strcat(featureName,'_P90'))=prctile(vector,90);

end
```

```
function TABLE_OUT = stats_table(stats)
%TABLE_OUT = stats_table(stats)
% Convert struct of statistics to a table
%
%   Inputs:
%           stats:      struct of statistics
%
%   Outputs:
%           TABLE_OUT:  table of observations x statsistics
```

```
%
%
% Author: Marcel Gonzalez Poy, 2016.

labels=fieldnames(stats);
feat_names=fieldnames(stats.(labels{1}));
numFEAT=numel(feat_names);
X=table;

for i=1:numFEAT
    X2=table;
    for j=1:length(labels);
        numOBS=length(stats.(labels{j}));
        for k=1:numOBS
            X2=[X2;struct2table(stats.(labels{j})(k).(feat_names{i}))];
        end
    end

    X=[X,X2];
end

Y=table;

for i=1:numel(labels)
    Y2=table;
    for j=1:numOBS
        Y2=[Y2;{i},labels(i)];
    end
    Y=[Y;Y2];
end

Y.Properties.VariableNames{1} = 'LABEL_NUM';
Y.Properties.VariableNames{2} = 'LABEL_NAME';

TABLE_OUT=[Y,X];
end
```

```
function [features,weights] = MI(features,labels,Q)
% function [features,weights] = MI(features,labels,Q) Estimates the mutual
% information between features and associated class labels using a
% quantized feature space.
%
%   Inputs:
%           features:       N x F sized matrix of features, where N is the
%           number of samples and F is the number of features labels:
%           N x 1 sized vector of class labels corresponding to each sample
%           Q:              the number of quantization levels used for the
%           features (default = 12)
%
%   Outputs:
%           features:       F x 1 sized vector of feature indices in the
%                              descending order of relevance.
%           weights:        F x 1 sized vector of feature relevances (MIs)
%           in the
%                              descending order.
%
% Author: Okko Rasanen, 2013. Mail: okko.rasanen@aalto.fi
%
% The algorithm can be freely used for research purposes.
%
% Please see J. Pohjalainen, O. Rasanen & S. Kadioglu: "Feature Selection
% Methods and Their Combinations in High-Dimensional Classification of
% Speaker Likability, Intelligibility and Personality Traits", Computer
% Speech and Language, 2015, for more details.

if nargin <3
    Q = 12;
end

edges = zeros(size(features,2),Q+1);

% Compute feature-specific quantization bins so that each bin has
% approximately equal number of samples in the training set
for k = 1:size(features,2)

    minval = min(features(:,k));
    maxval = max(features(:,k));
    if minval==maxval
        continue;
    end

    quantlevels = minval:(maxval-minval)/500:maxval;

    N = histc(features(:,k),quantlevels);
```

```
totsamples = size(features,1);

    N_cum = cumsum(N);

    edges(k,1) = -Inf;

    stepsize = totsamples/Q;

    for j = 1:Q-1
        a = find(N_cum > j.*stepsize,1);
        edges(k,j+1) = quantlevels(a);
    end

    edges(k,j+2) = Inf;
end

% Quantize data according to the obtained bins
S = zeros(size(features));
for k = 1:size(S,2)
    S(:,k) = quantize(features(:,k),edges(k,:))+1;
end

% Compute mutual information (MI) between the quantized features and the
% class labels
I = zeros(size(features,2),1);
for k = 1:size(features,2)
    I(k) = computeMI(S(:,k),labels,0);
end

% Sort features into descending order

[weights,features] = sort(I,'descend');


function [I,M,SP] = computeMI(seq1,seq2,lag)
% function [I,M,SP] = computeMI(seq1,seq2,lag) Computes the mutual
% information (MI) between seq1 and seq2 at the given delay (lag) between
% the sequences.
%
%   Inputs:
%
%       seq1:   a discrete sequence of length N seq2:   a discrete sequence
%       of length N lag:    the number of elements that seq1 is delayed
%       with respect to
%               seq2 (a positive or negative integer). Default = 0;


if nargin <3
    lag = 0;
end

if(length(seq1) ~= length(seq2))
    error('Input sequences are of different length');
end


% Count the frequency and probability of each symbol in seq1
lambda1 = max(seq1);
symbol_count1 = zeros(lambda1,1);

for k = 1:lambda1
    symbol_count1(k) = sum(seq1 == k);
end

symbol_prob1 = symbol_count1./sum(symbol_count1)+0.000001;


% Count the frequency and probability of each symbol in seq2
lambda2 = max(seq2);
symbol_count2 = zeros(lambda2,1);

for k = 1:lambda2
    symbol_count2(k) = sum(seq2 == k);
end

symbol_prob2 = symbol_count2./sum(symbol_count2)+0.000001;

% Compute the joint occurrence frequencies of symbol pairs at the given lag

M = zeros(lambda1,lambda2);
if(lag > 0)
    for k = 1:length(seq1)-lag
        loc1 = seq1(k);

        loc2 = seq2(k+lag);
```

```
        M(loc1,loc2) = M(loc1,loc2)+1;
    end
else
    for k = abs(lag)+1:length(seq1)
        loc1 = seq1(k);

        loc2 = seq2(k+lag);

        M(loc1,loc2) = M(loc1,loc2)+1;
    end
end

% Product of individual state probabilities as a matrix
SP = symbol_prob1*symbol_prob2';

% Pair joint probability
M = M./sum(M(:))+0.000001;

% Compute MI
I = sum(sum(M.*log2(M./SP)));

function y = quantize(x, q)
x = x(:);
nx = length(x);
nq = length(q);
y = sum(repmat(x,1,nq)>repmat(q,nx,1),2);

end
end
end
```

```
function [trainedClassifier, validationAccuracy , validationPredictions] =
trainClassifier(trainingData,...
    featuresSelected,labels)
% trainClassifier(trainingData)
%  returns a trained classifier and its accuracy. This code recreates the
%  classification model trained in Classification Learner app.
%
%   Input:
%       trainingData: the training data of same data type as imported
%        in the app (table or matrix).
%
%   Output:
%       trainedClassifier: a struct containing the trained classifier.
%        The struct contains various fields with information about the
%        trained classifier.
%
%       trainedClassifier.predictFcn: a function to make predictions
%        on new data. It takes an input of the same form as this training
%        code (table or matrix) and returns predictions for the response.
%        If you supply a matrix, include only the predictors columns (or
%        rows).
%
%       validationAccuracy: a double containing the accuracy in
%        percent. In the app, the History list displays this overall
%        accuracy score for each model.
%
%  Use the code to train the model with new data. To retrain your
%  classifier, call the function from the command line with your original
%  data or new data as the input argument trainingData.
%
%  For example, to retrain a classifier trained with the original data set
%  T, enter:
%    [trainedClassifier, validationAccuracy] = trainClassifier(T)
%
%  To make predictions with the returned 'trainedClassifier' on new data T,
%  use
%    yfit = trainedClassifier.predictFcn(T)
%
%  To automate training the same classifier with new data, or to learn how
%  to programmatically train classifiers, examine the generated code.

% Auto-generated by MATLAB on 06-May-2016 17:39:37

% ===============================================
%
% modified by Marcel Gonzalez Poy, 2016

%       new Inputs:
%           featuresSelected:       list of featutres in orded of relevance
%           labels:                 name of the classes
%       new Outputs:
%           validationAccuracy:     accuracy of the model
```

```
%           validationPrediction:   prediction done by kfoldPredict

% Extract predictors and response This code processes the data into the
% right shape for training the classifier.
inputTable = trainingData;
predictorNames = featuresSelected;
predictors = inputTable(:, predictorNames);
response = inputTable.LABEL_NAME;
isCategoricalPredictor = [false, false, false, false, false, false, false, false, false, false,...
    false, false, false, false, false, false, false, false, false, false, false, false, false,...
    false, false, false, false, false, false, false, false, false, false, false, false, false,...
    false, false, false, false, false, false, false, false, false, false, false, false, false,...
    false, false, false, false, false, false, false, false, false, false, false, false, false,...
    false, false, false, false, false, false, false, false, false, false, false, false, false,...
    false, false, false, false, false, false, false, false, false, false, false, false, false,...
    false, false, false, false, false, false, false, false];

% Train a classifier This code specifies all the classifier options and
% trains the classifier.
template = templateSVM(...
    'KernelFunction', 'polynomial', ...
    'PolynomialOrder', 2, ...
    'KernelScale', 'auto', ...
    'BoxConstraint', 1, ...
    'Standardize', true);
classificationSVM = fitcecoc(...
    predictors, ...
    response, ...
    'Learners', template, ...
    'Coding', 'onevsall', ...
    'ClassNames', labels);

% Create the result struct with predict function
predictorExtractionFcn = @(t) t(:, predictorNames);
svmPredictFcn = @(x) predict(classificationSVM, x);
trainedClassifier.predictFcn = @(x) svmPredictFcn(predictorExtractionFcn(x));

% Add additional fields to the result struct
trainedClassifier.RequiredVariables = featuresSelected;
trainedClassifier.ClassificationSVM = classificationSVM;
trainedClassifier.About = 'This struct is a trained classifier exported from Classification Learner
R2016a.';
trainedClassifier.HowToPredict = sprintf('To make predictions on a new table, T, use: \n  yfit =
c.predictFcn(T) \nreplacing ''c'' with the name of the variable that is this struct, e.g.
''trainedClassifier''. \n \nThe table, T, must contain the variables returned by: \n  c.RequiredVariables
\nVariable formats (e.g. matrix/vector, datatype) must match the original training data. \nAdditional
variables are ignored. \n \nFor more information, see <a href="matlab:helpview(fullfile(docroot,
''stats'', ''stats.map''), ''appclassification_exportmodeltoworkspace'')">How to predict using an
exported model</a>.');

% Perform cross-validation
partitionedModel = crossval(trainedClassifier.ClassificationSVM, 'KFold', 10);

% Compute validation accuracy
validationAccuracy = 1 - kfoldLoss(partitionedModel, 'LossFun', 'ClassifError');

% Compute validation predictions and scores
[validationPredictions, validationScores] = kfoldPredict(partitionedModel);
end
```

```
function [hImage, hText, hXText] = heatmap(mat, xlab, ylab, textmat, varargin)
% HEATMAP displays a matrix as a heatmap image
%
% USAGE: [hImage, hText, hTick] = heatmap(matrix, xlabels, ylabels,
% textmatrix, 'param', value, ...)
%
% INPUTS:
% * HEATMAP displays "matrix" as an image whose color intensities reflect
%   the magnitude of the values in "matrix".
%
% * "xlabels" (and "ylabels") can be either a numeric vector or cell array
%   of strings that represent the columns (or rows) of the matrix. If
%   either is not specified or empty, no labels will be drawn.
%
% * "textmat" can either be: 1 (or true), in which case the "matrix" values
% will be
%   displayed in each square, a format string, in which case the matrix
%   values will be displayed formatted according to the string specified, a
%   numeric matrix the size of "matrix", in which case those values will be
%   displayed as strings or a cell matrix of strings the size of "matrix",
%   in which case each string will be displayed. If not specified or empty,
%   no text will be displayed on the image
%
% OTHER PARAMETERS (passed as parameter-value pairs)
```

```
% * 'Colormap': Either a matrix of size numLevels-by-3 representing the
%   colormap to be used or a string or function handle representing a
%   function that returns a colormap, example, 'jet', 'hsv' or @cool.
%   Non-standard colormaps available within HEATMAP include 'money' and
%   'red'. By default, the current figure's colormap is used.
%
% * 'ColorLevels': The number of distinct levels in the colormap (default:
%   64). If more levels are specified than are present in the colormap, the
%   levels in the colormap are interpolated. If fewer are specified the
%   colormap is downsampled.
%
% * 'UseLogColormap': A true/false value which, if true, specifies that the
%   intensities displayed should match the log of the "matrix" values. Use
%   this if the data is naturally on a logarithmic scale (default: false)
%
% * 'UseFigureColormap': Specifies whether the figure's colormap should be
%   used. If false, the color intensities after applying the
%   specified/default colormap will be hardcoded, so that the image will be
%   independent of the figure's colormap. If this option is true, the
%   figure colormap in the end will be replaced by specified/default
%   colormap. (default = true)
%
% * 'NaNColor': A 3-element [R G B] vector specifying the color used to
% display NaN
%   or missing value. [0 0 0] corresponds to black and [1 1 1] to white. By
%   default MATLAB displays NaN values using the color assigned to the
%   lowest value in the colormap. Specifying this option automatically sets
%   the 'UseFigureColormap' option to false because the color mapping must
%   be computed prior to setting the nan color.
%
% * 'MinColorValue': A scalar number corresponding to the value of the data
%   that is mapped to the lowest color of the colormap. By default this is
%   the minimum value of the matrix input.
%
% * 'MaxColorValue': A scalar number corresponding to the value of the data
%   that is mapped to the highest color of the colormap. By default this is
%   the maximum value of the matrix input.
%
% * 'Parent': Handle to an axes object
%
% * 'TextColor': Either a color specification of all the text displayed on
%   the image or a string 'xor' which sets the EraseMode property of the
%   text objects to 'xor'. This will display all the text labels in a color
%   that contrasts its background.
%
% * 'FontSize': The initial fontSize of the text labels on the image. As
%   the image size is scaled the fontSize is shrunk appropriately.
%
% * 'ColorBar': Display colorbar. The corresponding value parameter should
%   be either logical 1 or 0 or a cell array of any additional parameters
%   you wish to pass to the colorbar function (such as location)
%
% * 'GridLines': Draw grid lines separating adjacent sections of the
%   heatmap. The value of the parameter is a LineStyle specification, for
%   example, :, -, -. or --. By default, no grid lines are drawn.
%
% * 'TickAngle': Angle of rotation of tick labels on x-axis. (Default: 0)
%
% * 'ShowAllTicks': Set to 1 or true to force all ticks and labels to be
%   drawn. This can make the axes labels look crowded. (Default: false)
%
% * 'TickFontSize': Font size of the X and Y tick labels. Default value is
%   the default axes font size, usually 10. Set to a lower value if many
%   tick labels are being displayed
%
% * 'TickTexInterpreter': Set to 1 or true to render tick labels using a
% TEX
%   interpreter. For example, '_b' and '^o' would be rendered as subscript
%   b and the degree symbol with the TEX interpreter. This parameter is
%   only available in MATLAB R2014b and above (Default: false)
%
% OUTPUTS:
% * hImage: handle to the image object
% * hText : handle to the text objects (empty if no text labels are drawn)
% * hTick : handle to the X-tick label text objects if tick angle is not 0
%           (empty otherwise)
%
% Notes:
% * The 'money' colormap displays a colormap where 0 values are mapped to
%   white, negative values displayed in varying shades of red and positive
%   values in varying shades of green
% * The 'red' colormap maps 0 values to white and higher values to red
%
% EXAMPLES: data = reshape(sort(randi(100,10)),10,10)-50; heatmap(data,
% cellstr(('A':'J')'), mean(data,2), '%0.0f%%',...
```

```
%         'Colormap', 'money', 'Colorbar', true, 'GridLines', ':',...
%         'TextColor', 'b')
% For detailed examples, see the associated document heatmap_examples.m

% Copyright The MathWorks, Inc. 2009-2014

% Handle missing inputs
if nargin < 1, error('Heatmap requires at least one input argument'); end
if nargin < 2, xlab = []; end
if nargin < 3, ylab = []; end
if nargin < 4, textmat = []; end

% Parse parameter/value inputs
p = parseInputs(mat, varargin{:});

% Get heatmap axes information if it already exists
p.axesInfo = getHeatmapAxesInfo(p.hAxes);

% Calculate the colormap based on inputs
p = calculateColormap(p, mat);

% Create heatmap image
p = plotHeatmap(p, mat); % New properties hImage and cdata added

% Generate grid lines if selected
generateGridLines(p);

% Set axes labels
[p, xlab, ylab, hXText, origPos] = setAxesTickLabels(p, xlab, ylab);

% Set text labels
[p, displayText, fontScaleFactor] = setTextLabels(p, mat, textmat);

% Add colorbar if selected
addColorbar(p, mat, textmat)

% Store heatmap properties in axes for callbacks
axesInfo = struct('Type', 'heatmap', 'Parameters', p, 'FontScaleFactor', ...
                  fontScaleFactor, 'mat', mat, 'hXText', hXText, ...
                  'origAxesPos', origPos);
axesInfo.xlab = xlab;
axesInfo.ylab = ylab;
axesInfo.displayText = displayText;
set(p.hAxes, 'UserData', axesInfo);

% Define callbacks
dObj = datacursormode(p.hFig);
set(dObj, 'Updatefcn', @cursorFun);

zObj = zoom(p.hFig);
set(zObj, 'ActionPostCallback', @(obj,evd)updateLabels(evd.Axes,true));

pObj = pan(p.hFig);
% set(pObj, 'ActionPreCallback',  @prePan);
set(pObj, 'ActionPostCallback', @(obj,evd)updateLabels(evd.Axes,true));

set(p.hFig, 'ResizeFcn', @resize)

% Set outputs
hImage = p.hImage;
hText = p.hText;

end

% ---------------------- Heatmap Creation Functions ----------------------

% Parse PV inputs & return structure of parameters
function param = parseInputs(mat, varargin)

p = inputParser;
p.addParamValue('Colormap',[]); %#ok<*NVREPL>
p.addParamValue('ColorLevels',[]);
p.addParamValue('TextColor',[0 0 0]);
p.addParamValue('UseFigureColormap',true);
p.addParamValue('UseLogColormap',false);
p.addParamValue('Parent',NaN);
p.addParamValue('FontSize',[]);
p.addParamValue('Colorbar',[]);
p.addParamValue('GridLines','none');
p.addParamValue('TickAngle',0);
p.addParamValue('ShowAllTicks',false);
p.addParamValue('TickFontSize',[]);
p.addParamValue('TickTexInterpreter',false);
p.addParamValue('NaNColor', [NaN NaN NaN], @(x)isnumeric(x) && length(x)==3 && all(x>=0) && all(x<=1));
p.addParamValue('MinColorValue', nan, @(x)isnumeric(x) && isscalar(x));
```

```
p.addParamValue('MaxColorValue', nan, @(x)isnumeric(x) && isscalar(x));
p.parse(varargin{:});

param = p.Results;

if ~ishandle(param.Parent) || ~strcmp(get(param.Parent,'type'), 'axes')
    param.Parent = gca;
end

ind = ~isinf(mat(:)) | isnan(mat(:));
if isnan(param.MinColorValue)
    param.MinColorValue = min(mat(ind));
end
if isnan(param.MaxColorValue)
    param.MaxColorValue = max(mat(ind));
end

% Add a few other parameters
param.hAxes = param.Parent;
param.hFig = ancestor(param.hAxes, 'figure');
param.IsGraphics2 = ~verLessThan('matlab','8.4');
param.ExplicitlyComputeImage = ~all(isnan(param.NaNColor)) ... NaNColor is specified
                        || ~param.IsGraphics2 && ~param.UseFigureColormap;


% if param.IsGraphics2 && ~param.UseFigureColormap &&
% ~isempty(param.ColorBar) % graphics v2
%     warning('heatmap:graphics2figurecolormap', 'The UseFigureColormap
%     false option with colorbar is not supported in versions R2014b and
%     above. In most such cases UseFigureColormap false is unnecessary');
% end


end

% Visualize heatmap image
function p = plotHeatmap(p, mat)

p.cdata = [];
if p.UseLogColormap
    p.Colormap = resamplecmap(p.Colormap, p.ColorLevels, ...
                          logspace(0,log10(p.ColorLevels),p.ColorLevels));
end

if p.ExplicitlyComputeImage
    % Calculate the color data explicitly and then display it as an image.
    n = p.MinColorValue;
    x = p.MaxColorValue;
    if x == n, x = n+1; end
    p.cdata = round((mat-n)/(x-n)*(p.ColorLevels-1)+1);
    %p.cdata = ceil((mat-n)/(x-n)*p.ColorLevels);
    p.cdata(p.cdata<1) = 1; % Clipping
    p.cdata(p.cdata>p.ColorLevels) = p.ColorLevels; % Clipping
    nanInd = find(isnan(p.cdata));
    p.cdata(isnan(p.cdata)) = 1;
    p.cdata = reshape(p.Colormap(p.cdata(:),:),[size(p.cdata) 3]);
    % Handle NaNColor case
    if ~all(isnan(p.NaNColor))
        p.cdata(nanInd                      ) = p.NaNColor(1); % Set red   color level of nan indices
        p.cdata(nanInd +   numel(p.cdata)/3) = p.NaNColor(2); % Set green color level of nan indices
        p.cdata(nanInd + 2*numel(p.cdata)/3) = p.NaNColor(3); % set blue  color level of nan indices
    end
    % Add a small dummy image so that colorbar subsequently works
    [indr, indc] = find(~isnan(mat),1);
    imagesc(indr, indc, mat(indr,indc),'Parent',p.hAxes);
    nextplot = get(p.hAxes,'nextplot');
    set(p.hAxes,'nextplot','add');
    p.hImage = image(p.cdata, 'Parent', p.hAxes);
    set(p.hAxes,'nextplot',nextplot);
    axis(p.hAxes,'tight');
else
    % Use a scaled image plot. Axes CLims and colormap will be set later
    p.hImage = imagesc(mat, 'Parent', p.hAxes);
end

set(p.hAxes, 'CLim', [p.MinColorValue p.MaxColorValue]); % Ensure proper clipping for colorbar
if p.UseFigureColormap
    set(p.hFig,'Colormap',p.Colormap);
elseif p.IsGraphics2
    % Set the axes colormap and limits
    colormap(p.hAxes, p.Colormap);
    %set(p.hAxes, 'CLim', [p.MinColorValue p.MaxColorValue]);
end
end
```

```
% Generate grid lines
function generateGridLines(p)
if ~strcmp(p.GridLines,'none')
    xlim = get(p.hAxes,'XLim');
    ylim = get(p.hAxes,'YLim');
    for i = 1:diff(xlim)-1
        line('Parent',p.hAxes,'XData',[i i]+.5, 'YData', ylim, 'LineStyle', p.GridLines);
    end
    for i = 1:diff(ylim)-1
        line('Parent',p.hAxes,'XData',xlim, 'YData', [i i]+.5, 'LineStyle', p.GridLines);
    end
end
end

% Add color bar
function addColorbar(p, mat, textmat)

if isempty(p.Colorbar)
    return;
elseif iscell(p.Colorbar)
    c = colorbar(p.Colorbar{:});
else
    c = colorbar;
end
if p.IsGraphics2
    c.Limits = p.hAxes.CLim;
    ticks = get(c,'Ticks');
else
    if p.ExplicitlyComputeImage || ~p.UseFigureColormap
        d = findobj(get(c,'Children'),'Tag','TMW_COLORBAR'); % Image
        set(d,'YData', get(p.hAxes,'CLim'));
        set(c,'YLim', get(p.hAxes,'CLim'));
    end
    ticks = get(c,'YTick');
    tickAxis = 'Y';
    if isempty(ticks)
        ticks = get(c,'XTick');
        tickAxis = 'X';
    end
end

if ~isempty(ticks)

    if ischar(textmat) % If format string, format colorbar ticks in the same way
        ticklabels = arrayfun(@(x){sprintf(textmat,x)},ticks);
    else
        ticklabels = num2str(ticks(:));
    end
    if p.IsGraphics2
        set(c, 'TickLabels', ticklabels);
    else
        set(c, [tickAxis 'TickLabel'], ticklabels);
    end

end


end


% ------------------------- Tick Label Functions -------------------------

% Set axes tick labels
function [p, xlab, ylab, hXText, origPos] = setAxesTickLabels(p, xlab, ylab)

if isempty(p.axesInfo) % Not previously a heatmap axes
    origPos = [get(p.hAxes,'Position') get(p.hAxes,'OuterPosition')];
else
    origPos = p.axesInfo.origAxesPos;
    set(p.hAxes, 'Position', origPos(1:4), 'OuterPosition', origPos(5:8));
end


if isempty(p.TickFontSize)
    p.TickFontSize = get(p.hAxes, 'FontSize');
else
    set(p.hAxes, 'FontSize', p.TickFontSize);
end

if isempty(ylab) % No ticks or labels
    set(p.hAxes,'YTick',[],'YTickLabel','');
else
    if isnumeric(ylab) % Numeric tick labels
        ylab = arrayfun(@(x){num2str(x)},ylab);
    end
```

```
    if ischar(ylab)
        ylab = cellstr(ylab);
    end
    ytick = get(p.hAxes, 'YTick');
    ytick(ytick<1|ytick>length(ylab)) = [];
    if p.ShowAllTicks || length(ytick) > length(ylab)
        ytick = 1:length(ylab);
    end
    set(p.hAxes,'YTick',ytick,'YTickLabel',ylab(ytick));
end
if p.IsGraphics2
    if p.TickTexInterpreter
        set(p.hAxes,'TickLabelInterpreter','tex');
    else
        set(p.hAxes,'TickLabelInterpreter','none');
    end
end
% Xlabels are trickier because they could have a TickAngle
hXText = []; % Default value
if isempty(xlab)
    set(p.hAxes,'XTick',[],'XTickLabel','');
else
    if isnumeric(xlab)
        xlab = arrayfun(@(x){num2str(x)},xlab);
    end
    if ischar(xlab)
        xlab = cellstr(xlab);
    end
    xtick = get(p.hAxes, 'XTick');
    xtick(xtick<1|xtick>length(xlab)) = [];
    if p.ShowAllTicks || length(xtick) > length(xlab)
        xtick = 1:length(xlab);
    end
    if p.IsGraphics2
        set(p.hAxes,'XTick',xtick,'XTickLabel',xlab(xtick),'XTickLabelRotation', p.TickAngle);
    else
        if p.TickAngle == 0
            set(p.hAxes,'XTick',xtick,'XTickLabel',xlab(xtick));
        else
            hXText = createXTicks(p.hAxes, p.TickAngle, xtick, xlab(xtick), p.TickTexInterpreter);
            adjustAxesToAccommodateTickLabels(p.hAxes, hXText);
        end
    end
end



end

% Create Rotated X Tick Labels (Graphics v1)
function hXText = createXTicks(hAxes, tickAngle, xticks, xticklabels, texInterpreter)

    axXLim = get(hAxes, 'XLim');
    [xPos, yPos] = calculateTextTickPositions(hAxes, axXLim, xticks);

    if texInterpreter
        interpreter = 'tex';
    else
        interpreter = 'none';
    end
    hXText = text(xPos, yPos, cellstr(xticklabels), 'Units', 'normalized', ...
                  'Parent', hAxes, 'FontSize', get(hAxes,'FontSize'), ...
                  'HorizontalAlignment', 'right', 'Rotation', tickAngle,...
                  'Interpreter', interpreter);

    set(hAxes, 'XTick', xticks, 'XTickLabel', '');

end

% Calculate positions of X tick text objects in normalized units
function [xPos, yPos] = calculateTextTickPositions(hAxes, xlim, ticks)
oldunits = get(hAxes,'Units');
set(hAxes,'units','pixels');
axPos = get(hAxes,'position');
set(hAxes,'units',oldunits);

xPos = (ticks - xlim(1))/diff(xlim);
%yPos = -.08 * ones(size(xPos));
yPos = -7.82/axPos(4) * ones(size(xPos));
end

% Adjust axes and tick positions so that everything fits well on screen
function adjustAxesToAccommodateTickLabels(hAxes, hXText)
% The challenge here is that the axes container, especially in a subplot is
% not well defined. The outer position property does not fully span or
```

```
% contain the x tick text objects. So here we just shrink the axes height
% just a little so that the axes and tick labels take the same room as the
% axes would have without the ticks.

[axPosP, axPosN, axOPP, axOPN, coPosP, textPosP]  = ...
    getGraphicsObjectsPositions(hAxes, hXText); %#ok<ASGLU>

header = axOPP(4) + axOPP(2) - axPosP(4) - axPosP(2); % Distance between top of axes and container
%in pixels;
delta = 5; % To adjust for overlap between area designated for regular ticks and area occupied by
%rotated ticks
axHeightP = axOPP(4) - header - delta - textPosP(4);

% Fudge axis position if labels are taking up too much room
if textPosP(4)/(textPosP(4)+axHeightP) > .7 % It's taking up more than 70% of total height
    axHeightP = (1/.7-1) * textPosP(4); % Minimum axis
end
axHeightN = max(0.0001, axHeightP / coPosP(4));

axPosN = axPosN + [0 axPosN(4)-axHeightN 0 axHeightN-axPosN(4)];
set(hAxes,'Position', axPosN)

end

% Calculate graphics objects positions in pixels and normalized units
function [axPosP, axPosN, axOPP, axOPN, coPosP, textPosP, textPosN] =...
         getGraphicsObjectsPositions(hAxes, hXText)

axPosN = get(hAxes, 'Position'); axOPN = get(hAxes, 'OuterPosition');
set(hAxes,'Units','Pixels');
axPosP = get(hAxes, 'Position'); axOPP = get(hAxes, 'OuterPosition');
set(hAxes,'Units','Normalized');

hContainer = get(hAxes,'Parent');
units = get(hContainer,'Units');
set(hContainer,'Units','Pixels');
coPosP = get(hContainer,'Position');
set(hContainer,'Units',units);

set(hXText,'Units','pixels'); % Measure height in pixels
extents = get(hXText,'Extent'); % Get heights for all text objects
extents = vertcat(extents{:}); % Collect heights in one matrix
textPosP = [min(extents(:,1)) min(extents(:,2)) ...
    max(extents(:,3)+extents(:,1))-min(extents(:,1)) ...
    max(extents(:,4))]; % Find dimensions of text label block
set(hXText,'Units','normalized'); % Restore previous behavior
extents = get(hXText,'Extent'); % Get heights for all text objects
extents = vertcat(extents{:}); % Collect heights in one matrix
textPosN = [min(extents(:,1)) min(extents(:,2)) ...
    max(extents(:,3)+extents(:,1))-min(extents(:,1)) ...
    max(extents(:,4))]; % Find dimensions of text label block

end


% -------------------------- Callback Functions -------------------------

% Update x-, y- and text-labels with respect to axes limits
function updateLabels(hAxes, axesLimitsChanged)

axInfo = getHeatmapAxesInfo(hAxes);
if isempty(axInfo), return; end
p = axInfo.Parameters;

% Update text font size to fill the square
if ~isempty(p.hText) && ishandle(p.hText(1))
    fs = axInfo.FontScaleFactor * getBestFontSize(hAxes);
    if fs > 0
        set(p.hText,'fontsize',fs,'visible','on');
    else
        set(p.hText,'visible','off');
    end
end

if axesLimitsChanged && ~isempty(axInfo.displayText) % If limits change & text labels are displayed
    % Get positions of text objects
    textPos = get(p.hText,'Position');
    textPos = vertcat(textPos{:});
    % Get axes limits
    axXLim = get(hAxes, 'XLim');
    axYLim = get(hAxes, 'YLim');
    % Find text objects within axes limit
    ind = textPos(:,1) > axXLim(1) & textPos(:,1) < axXLim(2) & ...
        textPos(:,2) > axYLim(1) & textPos(:,2) < axYLim(2);
    set(p.hText(ind), 'Visible', 'on');
```

```
    set(p.hText(~ind), 'Visible', 'off');
end

% Modify Y Tick Labels

if ~isempty(axInfo.ylab)
    axYLim = get(hAxes, 'YLim');
    if p.ShowAllTicks
        yticks = ceil(axYLim(1)):floor(axYLim(2));
    else
        set(hAxes, 'YTickMode', 'auto');
        yticks = get(hAxes, 'YTick');
        yticks = yticks( yticks == floor(yticks) );
        yticks(yticks<1|yticks>length(axInfo.ylab)) = [];
    end
    ylabels = repmat({''},1,max(yticks));
    ylabels(1:length(axInfo.ylab)) = axInfo.ylab;
    set(hAxes, 'YTick', yticks, 'YTickLabel', ylabels(yticks));
end

if ~isempty(axInfo.xlab)

    axXLim = get(hAxes, 'XLim');

    if p.ShowAllTicks
        xticks = ceil(axXLim(1)):floor(axXLim(2));
    else
        set(hAxes, 'XTickMode', 'auto');
        xticks = get(hAxes, 'XTick');
        xticks = xticks( xticks == floor(xticks) );
        xticks(xticks<1|xticks>length(axInfo.xlab)) = [];
    end
    xlabels = repmat({''},1,max(xticks));
    xlabels(1:length(axInfo.xlab)) = axInfo.xlab;

    if ~isempty(axInfo.hXText) % Rotated X tick labels exist
        try delete(axInfo.hXText); end %#ok<TRYNC>
        axInfo.hXText = createXTicks(hAxes, p.TickAngle, xticks, xlabels(xticks), p.TickTexInterpreter);
        set(hAxes, 'UserData', axInfo);
    else
        set(hAxes, 'XTick', xticks, 'XTickLabel', xlabels(xticks));
    end

    %adjustAxesToAccommodateTickLabels(hAxes, axInfo.hXText)
end

end

% Callback for data cursor
function output_txt = cursorFun(obj, eventdata)
hAxes = ancestor(eventdata.Target, 'axes');
axInfo = getHeatmapAxesInfo(hAxes);

pos = eventdata.Position;

if ~isempty(axInfo)

    try
        val = axInfo.displayText{pos(2), pos(1)};
    catch %#ok<CTCH>
        val = num2str(axInfo.mat(pos(2), pos(1)));
    end
    if isempty(axInfo.xlab), i = int2str(pos(1)); else i = axInfo.xlab{pos(1)}; end
    if isempty(axInfo.ylab), j = int2str(pos(2)); else j = axInfo.ylab{pos(2)}; end
    output_txt = sprintf('X: %s\nY: %s\nVal: %s', i, j, val);

else
    if length(pos) == 2
        output_txt = sprintf('X: %0.4g\nY: %0.4g', pos(1), pos(2));
    else
        output_txt = sprintf('X: %0.4g\nY: %0.4g\nZ: %0.4g', pos(1), pos(2), pos(3));
    end
end
end

% Callback for resize event
function resize(obj, evd)
hAxes = findobj(obj, 'type', 'axes');
for i = 1:length(hAxes)
    updateLabels(hAxes(i), false);
end
end

% Extract heatmap parameters for callback
function axInfo = getHeatmapAxesInfo(axH)
```

```
axInfo = get(axH, 'UserData');
try
    if ~strcmp(axInfo.Type, 'heatmap')
        axInfo = [];
    end
catch %#ok<CTCH>
    axInfo = [];
end
end


% ------------------------ Text Label Functions ------------------------

% Create text labels
function [p, displaytext, factor] = setTextLabels(p, mat, textmat)

if isempty(textmat)
    p.hText = [];
    displaytext = {};
    factor = 0;
    return
end

if isscalar(textmat) && textmat % If true convert mat to text
    displaytext = arrayfun(@(x){num2str(x)},mat);
elseif ischar(textmat) % If a format string, convert mat to text with specific format
    displaytext = arrayfun(@(x){sprintf(textmat,x)},mat);
elseif isnumeric(textmat) && numel(textmat)==numel(mat) % If numeric, convert to text
    displaytext = arrayfun(@(x){num2str(x)},textmat);
elseif iscellstr(textmat) && numel(textmat)==numel(mat)% Ifcell array of strings,it is already formatted
    displaytext = textmat;
else
    error('texmat is incorrectly specified');
end

if ischar(p.TextColor) && strcmp(p.TextColor,'xor')
    colorprop = 'EraseMode';
else
    colorprop = 'Color';
end

autoFontSize = getBestFontSize(p.hAxes);
if isempty(p.FontSize)
    p.FontSize = autoFontSize;
end
[xpos,ypos] = meshgrid(1:size(mat,2),1:size(mat,1));
if p.FontSize > 0
    p.hText = text(xpos(:),ypos(:),displaytext(:),'FontSize',p.FontSize,...
        'HorizontalAlignment','center', colorprop, p.TextColor,'Parent',p.hAxes);
else
    p.hText = text(xpos(:),ypos(:),displaytext(:),'Visible','off',...
        'HorizontalAlignment','center', colorprop, p.TextColor,'Parent',p.hAxes);
end

% Calculate factor to scale font size in future callbacks
factor = p.FontSize/autoFontSize;
if isnan(factor), factor = 1; end
if isinf(factor), factor = p.FontSize/6; end

% % Set up listeners to handle appropriate zooming
% addlistener(p.hAxes,{'XLim','YLim'},'PostSet',@(obj,evdata)resizeText);
% try
%     addlistener(p.hFig,'SizeChange',@(obj,evdata)resizeText);
% catch
%     addlistener(p.hFig,'Resize',@(obj,evdata)resizeText);
% end



%     function resizeText
%         if ~isempty(hText) && ishandle(hText(1))
%             fs = factor*getBestFontSize(hAxes); if fs > 0
%                 set(hText,'fontsize',fs,'visible','on');
%             else
%                 set(hText,'visible','off');
%             end
%         end
%     end

end

% Guess best font size from axes size using heuristics
function fs = getBestFontSize(imAxes)

hFig = ancestor(imAxes,'figure');
```

```
magicNumber = 80;
nrows = diff(get(imAxes,'YLim'));
ncols = diff(get(imAxes,'XLim'));
if ncols < magicNumber && nrows < magicNumber
    ratio = max(get(hFig,'Position').*[0 0 0 1])/max(nrows,ncols);
elseif ncols < magicNumber
    ratio = max(get(hFig,'Position').*[0 0 0 1])/ncols;
elseif nrows < magicNumber
    ratio = max(get(hFig,'Position').*[0 0 0 1])/nrows;
else
    ratio = 1;
end
fs = min(9,ceil(ratio/4));    % the gold formula
if fs < 4
    fs = 0;
end
end

% -------------------------- Colormap Functions --------------------------

% Determine the colormap to use
function p = calculateColormap(p, mat)

if isempty(p.Colormap)
    if p.IsGraphics2 && ~p.UseFigureColormap
        p.Colormap = colormap(p.hAxes);
    else
        p.Colormap = get(p.hFig,'Colormap');
    end
    if isempty(p.ColorLevels)
        p.ColorLevels = size(p.Colormap,1);
    else
        p.Colormap = resamplecmap(p.Colormap, p.ColorLevels);
    end
elseif ischar(p.Colormap) || isa(p.Colormap,'function_handle')
    if isempty(p.ColorLevels), p.ColorLevels = 64; end
    if strcmp(p.Colormap, 'money')
        p.Colormap = money(mat, p.ColorLevels);
    else
        p.Colormap = feval(p.Colormap,p.ColorLevels);
    end
elseif iscell(p.Colormap)
    p.Colormap = feval(p.Colormap{1}, p.Colormap{2:end});
    p.ColorLevels = size(p.Colormap,1);
elseif isnumeric(p.Colormap) && size(p.Colormap,2) == 3
    p.ColorLevels = size(p.Colormap,1);
else
    error('Incorrect value for colormap parameter');
end % p.Colormap is now a p.ColorLevels-by-3 rgb vector
assert(p.ColorLevels == size(p.Colormap,1));

end

% Resample a colormap by interpolation or decimation
function cmap = resamplecmap(cmap, clevels, xi)

t = cmap;
if nargin < 3
    xi = linspace(1,clevels,size(t,1));
end
xi([1 end]) = [1 clevels]; % These need to be exact for the interpolation to
% work and we don't want machine precision messing it up
cmap = [interp1(xi, t(:,1), 1:clevels);...
        interp1(xi, t(:,2), 1:clevels);...
        interp1(xi, t(:,3), 1:clevels)]';
end

% Generate Red-White-Green color map
function cmap = money(data, clevels)
% Function to make the heatmap have the green, white and red effect
n = min(data(:));
x = max(data(:));
if x == n, x = n+1; end
zeroInd = round(-n/(x-n)*(clevels-1)+1);
if zeroInd <= 1 % Just green
    b = interp1([1 clevels], [1 0], 1:clevels);
    g = interp1([1 clevels], [1 1], 1:clevels);
    r = interp1([1 clevels], [1 0], 1:clevels);
elseif zeroInd >= clevels, % Just red
    b = interp1([1 clevels], [0 1], 1:clevels);
    g = interp1([1 clevels], [0 1], 1:clevels);
    r = interp1([1 clevels], [1 1], 1:clevels);
else
    b = interp1([1 zeroInd clevels], [0 1 0], 1:clevels);
    g = interp1([1 zeroInd clevels], [0 1 1], 1:clevels);
```

```
    r = interp1([1 zeroInd clevels], [1 1 0], 1:clevels);
end

cmap = [r' g' b'];
end

% Generate Red-White color map
function cmap = red(levels)
r = ones(levels, 1);
g = linspace(1, 0, levels)';
cmap = [r g g];
end
```

```
function varargout = CTSSER_GUI(varargin)
%CTSSER_GUI MATLAB code file for CTSSER_GUI.fig
%      CTSSER_GUI, by itself, creates a new CTSSER_GUI or raises the existing
%      singleton*.
%
%      H = CTSSER_GUI returns the handle to a new CTSSER_GUI or the handle to
%      the existing singleton*.
%
%      CTSSER_GUI('Property','Value',...) creates a new CTSSER_GUI using the
%      given property value pairs. Unrecognized properties are passed via
%      varargin to CTSSER_GUI_OpeningFcn.  This calling syntax produces a
%      warning when there is an existing singleton*.
%
%      CTSSER_GUI('CALLBACK') and CTSSER_GUI('CALLBACK',hObject,...) call the
%      local function named CALLBACK in CTSSER_GUI.M with the given input
%      arguments.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help CTSSER_GUI

% Last Modified by GUIDE v2.5 02-Jun-2016 23:36:41

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @CTSSER_GUI_OpeningFcn, ...
                   'gui_OutputFcn',  @CTSSER_GUI_OutputFcn, ...
                   'gui_LayoutFcn',  [], ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before CTSSER_GUI is made visible.
function CTSSER_GUI_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   unrecognized PropertyName/PropertyValue pairs from the
%            command line (see VARARGIN)


%set default handles
handles.main_dir='';
handles.fs=22050;
handles.frame_size=[0.023 ,0.046, 0.092];
handles.win_size=handles.frame_size(1);
handles.HPF_fc=100;
set(handles.cutoffHz,'String',handles.HPF_fc);
set(handles.frameS1,'String',handles.frame_size(1));
set(handles.frameS2,'String',handles.frame_size(2));
set(handles.frameS3,'String',handles.frame_size(3));

%create Models directory
[~,s]=fileattrib;
handles.ModelsDir=strcat(s.Name, '\CTSSER_Models');
if(~exist(handles.ModelsDir,'dir')),
```

```
mkdir(handles.ModelsDir);
end
addpath(genpath(handles.ModelsDir));
%enable prediction if exist a model
if (length(dir(handles.ModelsDir))>2)
        set(handles.pushbutton_SelFile,'Enable','on');
end


% Choose default command line output for CTSSER_GUI
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes CTSSER_GUI wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = CTSSER_GUI_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in pushbutton_selDir.
function pushbutton_selDir_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_selDir (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

handles.main_dir=uigetdir;
set(handles.text_mainDir,'String',handles.main_dir);
if(handles.main_dir==0)
    helpdlg('Please select directory');
    set(handles.startTrain,'Enable','off');
else
    set(handles.startTrain,'Enable','on');
end

% Save the handles structure.
guidata(hObject,handles)

% --- Executes on selection change in popupmenu_fs.
function popupmenu_fs_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu_fs (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu_fs contents as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu_fs
% Determine the selected data set.
str = get(hObject,'String');
val = get(hObject,'Value');
% Set current data to the selected data set.
switch str{val};
case '16000'
   handles.fs = 16000;
   handles.frame_size=[0.032 , 0.064, 0.128];
case '22050'
   handles.fs = 22050;
   handles.frame_size=[0.023 , 0.046, 0.092];
case '44100'
   handles.fs=  44100;
    handles.frame_size=[0.023 , 0.046, 0.092];
case '48000'
   handles.fs = 48000;
    handles.frame_size=[0.021 , 0.042, 0.085];
end

set(handles.frameS1,'String',handles.frame_size(1));
set(handles.frameS2,'String',handles.frame_size(2));
set(handles.frameS3,'String',handles.frame_size(3));
% Save the handles structure.
guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function popupmenu_fs_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu_fs (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on slider movement.
function cutoffSlider_Callback(hObject, eventdata, handles)
% hObject    handle to cutoffSlider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider

fc=get(hObject,'Value');
handles.HPF_fc=round(fc);
set(handles.cutoffHz,'String',round(handles.HPF_fc));
% Save the handles structure.
guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function cutoffSlider_CreateFcn(hObject, eventdata, handles)
% hObject    handle to cutoffSlider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on button press in radiobutton_fSize1.
function radiobutton_fSize1_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton_fSize1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton_fSize1
handles.win_Size=handles.frame_size(1);
% Save the handles structure.
guidata(hObject,handles)


% --- Executes on button press in radiobutton_fSize2.
function radiobutton_fSize2_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton_fSize2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton_fSize2
handles.win_Size=handles.frame_size(2);
% Save the handles structure.
guidata(hObject,handles)


% --- Executes on button press in radiobutton_fSize3.
function radiobutton_fSize3_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton_fSize3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton_fSize3
handles.win_Size=handles.frame_size(3);
% Save the handles structure.
guidata(hObject,handles)


% --- Executes on button press in startTrain.
function startTrain_Callback(hObject, eventdata, handles)
% hObject    handle to startTrain (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%disable training
set(handles.startTrain,'Enable','off')
set(handles.pushbutton_selDir,'Enable','off');
set(handles.text_mainDir,'Enable','off');
set(handles.popupmenu_fs,'Enable','off');
set(handles.radiobutton_fSize1,'Enable','off');
set(handles.radiobutton_fSize2,'Enable','off');
set(handles.radiobutton_fSize3,'Enable','off');
```

```
set(handles.frameS1,'Enable','off');
set(handles.frameS2,'Enable','off');
set(handles.frameS3,'Enable','off');
set(handles.cutoffSlider,'Enable','off');
set(handles.cutoffHz,'Enable','off');

%disable predict
set(handles.startPredict,'Enable','off')
set(handles.pushbutton_SelFile,'Enable','off');
set(handles.text_file,'Enable','off');
set(handles.modelsList,'Enable','off');
set(handles.dispSR,'Enable','off');
set(handles.dispFS,'Enable','off');
set(handles.dispHPF,'Enable','off');
set(handles.dispACC,'Enable','off');
set(handles.dispClasses,'Enable','off');
set(handles.dispParam,'Enable','off');
set(handles.dispConfM,'Enable','off');
set(handles.dispTfeat,'Enable','off');

%open/create log.txt to print proces steps in listBox CTSout
handles.fid=fopen('log.txt','w+');
fprintf(handles.fid,'--->Type CTRL+C to abort process\n!!!It will cause an error in log.txt!!!<--fclose
to solve\n\n');
CTS_listboxDraw(handles.fid,handles.CTSout);
%execute SER_TRAINING
[model,accuracy]=SER_TRAINING(handles.main_dir,handles.fs,handles.win_Size,handles.HPF_fc,handles.fid,han
dles.CTSout);

%Create folder to save models and confusion matrix figure
flipdir=flip(handles.main_dir);
modelName=strcat(flip( flipdir(1:strfind(flipdir,'\')-1)),'_',int2str(model.opts.fs),'_',...
    int2str(model.opts.frame_size*1000),'_',int2str(model.opts.HPF_fc));

%subdirectory for each model created
subdirModel=strcat(handles.ModelsDir,'\',modelName);
if(~exist(subdirModel,'dir')),
    mkdir(subdirModel);
end
addpath(genpath(subdirModel));
%save model and confusion matrix
filepath=strcat(subdirModel,'\',modelName);
save(filepath,'model');
saveas(gcf, filepath,'png');

%dispaly success on CTSout
fprintf(handles.fid,'\n*****MODEL CREATED SUCCESSFULLY****\nNAME: %s\nACCURACY: %.2f%
%',modelName,accuracy*100);
CTS_listboxDraw(handles.fid,handles.CTSout);
fclose(handles.fid);

%enable
set(handles.startTrain,'Enable','on')
set(handles.pushbutton_selDir,'Enable','on');
set(handles.text_mainDir,'Enable','on');
set(handles.popupmenu_fs,'Enable','on');
set(handles.radiobutton_fSize1,'Enable','on');
set(handles.radiobutton_fSize2,'Enable','on');
set(handles.radiobutton_fSize3,'Enable','on');
set(handles.frameS1,'Enable','on');
set(handles.frameS2,'Enable','on');
set(handles.frameS3,'Enable','on');
set(handles.cutoffSlider,'Enable','on');
set(handles.cutoffHz,'Enable','on');

%enable Predict
set(handles.pushbutton_SelFile,'Enable','on');
set(handles.text_file,'Enable','on');
set(handles.modelsList,'Enable','on');
set(handles.dispSR,'Enable','on');
set(handles.dispFS,'Enable','on');
set(handles.dispHPF,'Enable','on');
set(handles.dispACC,'Enable','on');
set(handles.dispClasses,'Enable','on');
set(handles.dispParam,'Enable','on');
set(handles.dispConfM,'Enable','on');
set(handles.dispTfeat,'Enable','on');

% Save the handles structure.
guidata(hObject,handles)


% --- Executes on button press in startPredict.
function startPredict_Callback(hObject, eventdata, handles)
% hObject    handle to startPredict (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%disable training
set(handles.pushbutton_selDir,'Enable','off');
set(handles.text_mainDir,'Enable','off');
set(handles.popupmenu_fs,'Enable','off');
set(handles.radiobutton_fSize1,'Enable','off');
set(handles.radiobutton_fSize2,'Enable','off');
set(handles.radiobutton_fSize3,'Enable','off');
set(handles.frameS1,'Enable','off');
set(handles.frameS2,'Enable','off');
set(handles.frameS3,'Enable','off');
set(handles.cutoffSlider,'Enable','off');
set(handles.cutoffHz,'Enable','off');

%disable predict
set(handles.startPredict,'Enable','off')
set(handles.pushbutton_SelFile,'Enable','off');
set(handles.text_file,'Enable','off');
set(handles.modelsList,'Enable','off');
set(handles.dispSR,'Enable','off');
set(handles.dispFS,'Enable','off');
set(handles.dispHPF,'Enable','off');
set(handles.dispACC,'Enable','off');
set(handles.dispClasses,'Enable','off');
set(handles.dispParam,'Enable','off');
set(handles.dispConfM,'Enable','off');
set(handles.dispTfeat,'Enable','off');

prediction=SER_PREDICT(handles.file,handles.currentModel);

set(handles.text_pred,'String',prediction);

%enable training
set(handles.pushbutton_selDir,'Enable','on');
set(handles.text_mainDir,'Enable','on');
set(handles.popupmenu_fs,'Enable','on');
set(handles.radiobutton_fSize1,'Enable','on');
set(handles.radiobutton_fSize2,'Enable','on');
set(handles.radiobutton_fSize3,'Enable','on');
set(handles.frameS1,'Enable','on');
set(handles.frameS2,'Enable','on');
set(handles.frameS3,'Enable','on');
set(handles.cutoffSlider,'Enable','on');
set(handles.cutoffHz,'Enable','on');

%enable predict
set(handles.startPredict,'Enable','on')
set(handles.pushbutton_SelFile,'Enable','on');
set(handles.text_file,'Enable','on');
set(handles.modelsList,'Enable','on');
set(handles.dispSR,'Enable','on');
set(handles.dispFS,'Enable','on');
set(handles.dispHPF,'Enable','on');
set(handles.dispACC,'Enable','on');
set(handles.dispClasses,'Enable','on');
set(handles.dispParam,'Enable','on');
set(handles.dispConfM,'Enable','on');
set(handles.dispTfeat,'Enable','on');

guidata(hObject,handles);

% --- Executes on button press in pushbutton_SelFile.
function pushbutton_SelFile_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton_SelFile (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

[handles.file, handles.filePath]=uigetfile('*.wav');
set(handles.text_file,'String',handles.file);

if(handles.file==0)
    helpdlg('Please select file');
    %disable predict
    set(handles.startPredict,'Enable','off')
    set(handles.text_file,'Enable','off');
    set(handles.modelsList,'Enable','off');
    set(handles.dispSR,'Enable','off');
    set(handles.dispFS,'Enable','off');
    set(handles.dispHPF,'Enable','off');
    set(handles.dispACC,'Enable','off');
    set(handles.dispClasses,'Enable','off');
    set(handles.dispParam,'Enable','off');
    set(handles.dispConfM,'Enable','off');
```

```
    set(handles.dispTfeat,'Enable','off');
else
    addpath(genpath(handles.filePath));
    set(handles.modelsList,'Enable','on');
    set(handles.text_file,'Enable','on');
    handles.list='';
    i=0;
    D=dir(handles.ModelsDir);
    for n=1:length(D)
        if(strcmp('..',D(n).name)==0 && strcmp('.',D(n).name)==0)
            i=i+1;
            handles.list{i}=D(n).name;
        end
    end
    if (~isempty(handles.list))
        set(handles.modelsList,'String',handles.list)
    end
end

% Save the handles structure.
guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function pushbutton_SelFile_CreateFcn(hObject, eventdata, handles)
% hObject    handle to pushbutton_SelFile (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% --- Executes on selection change in modelsList.
function modelsList_Callback(hObject, eventdata, handles)
% hObject    handle to modelsList (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns modelsList contents as cell array
%        contents{get(hObject,'Value')} returns selected item from modelsList

%Refresh list of models
handles.list='';
i=0;
D=dir(handles.ModelsDir);
for n=1:length(D)
    if(strcmp('..',D(n).name)==0 && strcmp('.',D(n).name)==0)
        i=i+1;
        handles.list{i}=D(n).name;
    end
end
if (~isempty(handles.list))
    set(handles.modelsList,'String',handles.list)
end
%Get model and confusion Matrix image
handles.mdl=handles.list{get(hObject,'Value')};
mdlroot=strcat(handles.ModelsDir,'\',handles.mdl,'\',handles.mdl);
structMdl=load(mdlroot,'-mat');
handles.confM_image=imread(strcat(mdlroot,'.png'));
handles.currentModel=structMdl.model;

%enable display parameters
set(handles.dispSR,'Enable','on');
set(handles.dispFS,'Enable','on');
set(handles.dispHPF,'Enable','on');
set(handles.dispACC,'Enable','on');
set(handles.dispClasses,'Enable','on');
set(handles.dispParam,'Enable','on');
set(handles.dispConfM,'Enable','on');
set(handles.dispTfeat,'Enable','on');

%set displays
set(handles.dispSR,'String',handles.currentModel.opts.fs);
set(handles.dispFS,'String',handles.currentModel.opts.frame_size);
set(handles.dispHPF,'String',handles.currentModel.opts.HPF_fc);
set(handles.dispACC,'String',round(handles.currentModel.Accu*100,2));
set(handles.dispClasses,'String',handles.currentModel.ClassificationSVM.ClassNames);
set(handles.dispParam,'String',handles.currentModel.RequiredVariables);
set(handles.dispTfeat,'String',numel(handles.currentModel.RequiredVariables));

%enable start prediction
set(handles.startPredict,'Enable','on');

guidata(hObject,handles);


% --- Executes on button press in dispConfM.
function dispConfM_Callback(hObject, eventdata, handles)
% hObject    handle to dispConfM (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

figName=strcat('Confusion Matrix: ',handles.mdl);
figure('Name',figName), imshow(handles.confM_image);

% --- Executes during object creation, after setting all properties.
function modelsList_CreateFcn(hObject, eventdata, handles)
% hObject    handle to modelsList (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes during object creation, after setting all properties.
function dispSR_CreateFcn(hObject, eventdata, handles)
% hObject    handle to dispSR (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --- Executes during object creation, after setting all properties.
function dispFS_CreateFcn(hObject, eventdata, handles)
% hObject    handle to dispFS (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --- Executes during object creation, after setting all properties.
function dispHPF_CreateFcn(hObject, eventdata, handles)
% hObject    handle to dispHPF (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --- Executes during object creation, after setting all properties.
function dispACC_CreateFcn(hObject, eventdata, handles)
% hObject    handle to dispACC (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --- Executes on selection change in dispParam.
function dispParam_Callback(hObject, eventdata, handles)
% hObject    handle to dispParam (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns dispParam contents as cell array
%        contents{get(hObject,'Value')} returns selected item from dispParam


% --- Executes during object creation, after setting all properties.
function dispParam_CreateFcn(hObject, eventdata, handles)
% hObject    handle to dispParam (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function CTSout_CreateFcn(hObject, eventdata, handles)
% hObject    handle to CTSout (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function CTS_listboxDraw(fid,CTSout)
%%CTS_listboxDraw(fid,CTSout)
```

```
% Print the proces in a listbox of CTSSER GUI.
%
%   Inputs:
%           fid:        file ID
%           CTSout:     the handles of GUI where printing the log
%
% Author: Marcel Gonzalez Poy, 2016.

frewind(fid);
set(CTSout,'String',fscanf(fid,'%c'));
drawnow;

end
```