# ACA Project: Speech Emotion Detection

In [1]:
```
import numpy as np
import librosa
from scipy.io import wavfile
import os, time, csv, datetime

from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
```

In [2]:
```
## read audio wav from dataset TESS and RAVDESS

parameters = [7, -1, 1024, 768, 80, 300, 8000, 50]
[emo_read_num, file_read_num, win_size, hop_size, min_freq, max_fund_f
req, max_freq, mfcc_size] = [int(x) for x in parameters]

TESS_trim = 0.62
RAVDESS_trim = 0.26
magic = 43195

time_very_start = time.time()
print('Start')

features = []
labels = []
nowdate = datetime.datetime.now()
savename = 'feat_'  + str(win_size) + 'win_' + \
           '[' + str(nowdate.month).zfill(2) + str(nowdate.day).zfill(
2) + '-' + \
           str(nowdate.hour).zfill(2) + str(nowdate.minute).zfill(2) +
'].csv'

parent_dir = os.path.dirname(os.getcwd())

for dataset in ['TESS', 'RAVDESS']:
    dataset_dir = os.path.join(parent_dir, 'project', dataset)
    for emotion in range(7):
        if emotion >= emo_read_num:
            break
        time_start = time.time()
        print('Reading emotion #' + str(emotion) + ' in ' + dataset +
'...')
        emotion_dir = os.path.join(dataset_dir, str(emotion))
```

```
        file_count = 0
        file_list = os.listdir(emotion_dir)
        for file in file_list:
            if file_read_num != -1 and file_count >= file_read_num:
                break
            if (not file.endswith('.wav')) or file[0] == '.':
                continue
            fs, x = wavfile.read(os.path.join(emotion_dir, file))

            if len(x) >= magic:
                if dataset == 'TESS':
                    x = x[int((len(x) - magic) / 2):][:magic]
                else:
                    x = x[-magic:]
            else: # add zeros at end
                x = np.concatenate((x, [0] * (magic - len(x))))

            x = x / 32768 # convert 16-bit PCM to [-1, 1]

            s = librosa.feature.melspectrogram(y=x, sr=fs, n_fft=win_s
ize, hop_length=hop_size)
            mfcc = librosa.feature.mfcc(S=librosa.power_to_db(s), sr=f
s, n_mfcc=mfcc_size)

            rms = librosa.feature.rmse(y=x, frame_length=win_size, hop
_length=hop_size)
            zcr = librosa.feature.zero_crossing_rate(y=x, frame_length
=win_size, hop_length=hop_size)
            centroid = librosa.feature.spectral_centroid(y=x, sr=fs, n
_fft=win_size, hop_length=hop_size)

            # pitch
            min_lag = int(fs / max_fund_freq)
            max_lag = int(fs / min_freq)
            L = range(min_lag, max_lag + 1)
            spec = librosa.core.stft(x, n_fft=win_size, hop_length=hop
_size, win_length=win_size)
            dividend = np.transpose([np.real(np.fft.ifft(row)) for row
in (np.absolute(spec) ** 2).transpose()])
            divisor = np.transpose([win_size - lag + 1 for lag in L])
            acf = dividend[L] / divisor[:, None]
            i_max = np.argmax(acf, axis=0)
            pitch = fs / (i_max - 1 + min_lag)

            if len(set([len(mfcc[0]), len(rms[0]), len(zcr[0]), len(ce
ntroid[0]), len(pitch)])) != 1:
                print('  Error: File ' + file + ' has different number
s of windows among different features!')
                continue
```

```
            if dataset == 'TESS':
                gender = np.vstack(([0] * len(rms[0]), [1] * len(rms[0
]))) 
            else:
                if int(file[19]) % 2 == 0:
                    gender = np.vstack(([0] * len(rms[0]), [1] * len(r
ms[0]))) # female
                else:
                    gender = np.vstack(([1] * len(rms[0]), [0] * len(r
ms[0]))) # male

            # vertically concatenate features of all windows
            concat = np.vstack((mfcc, rms, zcr, centroid, pitch, gende
r))
            features.append(concat)
            labels.append(emotion)
            file_count += 1
#         print('    ' + str(file_count) + ' files feature extracted.
(' + str(int(time.time() - time_start)) + ' s)')

print('Finished. (' + str(int(time.time() - time_very_start)) + ' s in
total)')
```

```
Start
Reading emotion #0 in TESS...
Reading emotion #1 in TESS...
Reading emotion #2 in TESS...
Reading emotion #3 in TESS...
Reading emotion #4 in TESS...
Reading emotion #5 in TESS...
Reading emotion #6 in TESS...
Reading emotion #0 in RAVDESS...
Reading emotion #1 in RAVDESS...
Reading emotion #2 in RAVDESS...
Reading emotion #3 in RAVDESS...
Reading emotion #4 in RAVDESS...
Reading emotion #5 in RAVDESS...
Reading emotion #6 in RAVDESS...
Finished. (22 s in total)
```

In [3]:
```
fea = np.array(features)
lab = np.array(labels)
print(fea.shape)
print(lab.shape)
rowdim = fea.shape[0]
ydim = fea.shape[1]
xdim = fea.shape[2]
# print(ydim)
```

```
(2618, 56, 57)
(2618,)
```

In [4]:
```
## original used to read in data, not use anymore

# import pandas as pd
# import numpy as np

# # read data from xls file
# df = pd.read_csv('features.csv', header = None, na_values = '?', ind
ex_col = None)

# # filenames = ['features2.csv', 'features3.csv', 'features4.csv', 'f
eatures5.csv', 'features6.csv','test1.csv','test2.csv']
# # for filename in filenames:
# #     df1 = pd.read_csv(filename, header = None, na_values = '0', in
dex_col = None)
# #     df = pd.concat([df,df1], ignore_index = True)
# data_column = np.shape(df)[1]
# data_row = np.shape(df)[0]
# print(np.shape(df))
# df.head(5)

# # get the value
# y_raw = np.array(df[data_column-2])

# # list of 1D-features in the order of MFCC, Energy, Pitch
# X_raw = np.array(pd.DataFrame(df, columns = df.columns[0:(data_colum
n-2)]))

# print(np.shape(X_raw))
# print(np.shape(y_raw))
# # print(X_raw)
# # print(y_raw)
```

In [5]:
```
# preprocessing the data
X_raw = fea.reshape(rowdim, ydim*xdim)
y_raw = lab
print(X_raw.shape)
print(y_raw.shape)

# normalize
X = preprocessing.scale(X_raw)
```

```
(2618, 3192)
(2618,)
```

In [6]:
```
# randomly split data into train(80%) and test(20%) set
test_rate = 0.2
Xtr, Xts, ytr, yts = train_test_split(X, y_raw, test_size=test_rate, r
andom_state=0)
print(Xtr.shape)
print(ytr.shape)
print(Xts.shape)
print(yts.shape)
```

```
(2094, 3192)
(2094,)
(524, 3192)
(524,)
```

# Classifier 1: SVM model

In [7]:
```
from sklearn import svm
from sklearn.metrics import accuracy_score, confusion_matrix

# used for finding best parameters(C and gamma)
# for i in range(1,10):
#     svc = svm.SVC(kernel = 'rbf', C = i/10, gamma = 'auto', verbose
= 10)
#     svc.fit(Xtr,ytr)
#     yhat_ts = svc.predict(Xts)
#     acc = accuracy_score(yhat_ts,yts)
#     print(acc)

svc = svm.SVC(kernel = 'rbf', C = 6, gamma = 'auto', verbose = 10)
svc.fit(Xtr,ytr)
yhat_ts = svc.predict(Xts)
acc = accuracy_score(yhat_ts,yts)
print(acc)
```

```
[LibSVM]0.851145038168
```

In [8]:
```
# confusion matrix
size0 = np.shape(np.where(yts==0))[1]
size1 = np.shape(np.where(yts==1))[1]
size2 = np.shape(np.where(yts==2))[1]
size3 = np.shape(np.where(yts==3))[1]
size4 = np.shape(np.where(yts==4))[1]
size5 = np.shape(np.where(yts==5))[1]
size6 = np.shape(np.where(yts==6))[1]
size = np.array([1/size0, 1/size1, 1/size2, 1/size3, 1/size4, 1/size5,
1/size6])

dim = yts.shape[0]
# print(dim)
C = confusion_matrix(yts, yhat_ts, labels=None, sample_weight=None)
# print(C)
C_normalized = size*C
print(np.array_str(C_normalized, precision=4))
```

```
[[ 0.9429  0.      0.      0.      0.0128  0.023   0.0135]
 [ 0.0143  0.8846  0.0125  0.      0.0513  0.      0.    ]
 [ 0.0143  0.      0.8375  0.0482  0.0256  0.023   0.0541]
 [ 0.0286  0.0192  0.0375  0.8072  0.0256  0.0345  0.0676]
 [ 0.0143  0.0577  0.025   0.0964  0.8205  0.      0.    ]
 [ 0.0571  0.      0.025   0.012   0.      0.9195  0.    ]
 [ 0.0429  0.0192  0.05    0.0241  0.0641  0.0345  0.7568]]
```

In [9]:
```
import keras
from keras import applications
from keras.preprocessing.image import ImageDataGenerator
from keras import optimizers
from keras.models import Sequential, Model
from keras.layers import Dropout, Flatten, Dense, Input, Convolution2D
, MaxPooling2D, Activation, concatenate, LSTM
from keras.layers.normalization import BatchNormalization
```

```
Using TensorFlow backend.
```

In [10]:
```
import keras.backend as K
K.clear_session()
```

In [11]:
```
## use pre-trained deep learning network vgg16
# not useful in this task, bad performance
```

In [12]:
```
# # pre-trained deep learning network vgg16
# Xtr = Xtr.reshape(Xtr.shape[0], ydim, xdim, 1)
# Xts = Xts.reshape(Xts.shape[0], ydim, xdim, 1)
# print(Xtr.shape)
# print(Xts.shape)
# Xtr_1 = []
# Xts_1 = []
# Xtr_1 = Xtr.repeat(3, axis=3)
# Xts_1 = Xts.repeat(3, axis=3)
# print(Xtr_1.shape)
# print(Xts_1.shape)

# pre_trained = 'vgg16'

# # Load appropriate packages
# from keras.applications.vgg16 import VGG16
# from keras.applications.vgg16 import decode_predictions, preprocess_
input

# input_shape = (ydim,xdim,3)
# base_model = applications.VGG16(weights='imagenet', include_top = Fa
lse, input_shape = input_shape)
```

In [13]:
```
# model = Sequential()

# for layers in base_model.layers:
#     model.add(layers)

# for layers in model.layers:
#     layers.trainable = False

# model.add(Flatten())
# model.add(Dense(256,activation = 'relu'))
# model.add(Dropout(0.5))
# model.add(Dense(7, activation = 'sigmoid'))
# model.summary()
```

In [14]:
```
# opt = optimizers.Adam(lr=0.001) # beta_1=0.9, beta_2=0.999, epsilon=
1e-08, decay=0.0)
# model.compile(optimizer=opt,
#               loss='sparse_categorical_crossentropy',
#               metrics=['accuracy'])
```

In [15]:
```
# nepochs = 5  # Number of epochs

# # Call the fit function
# model.fit(Xtr_1, ytr, batch_size=32, epochs=nepochs)
```

In [16]:
```
# yhat = model.predict(Xts_1)
# yhat = np.argmax(yhat, axis=1)
# print(yhat.shape)
# yhat
```

In [17]:
```
# accuracy_score(yts, yhat)
```

In [18]:
```
# model.compile(loss='categorical_crossentropy', optimizer=Adam)
# model.fit(x, y, batch_size=batch_size, nb_epoch=nb_epoch,
#                         verbose=2, validation_data=(xt, yt), show_acc
uracy=True)
```

# important notice

There are 3 models in the code below, one time can only run one model, uncomment/comment use '#' as you want. Keyboard shortcut for uncomment/comment: 'command' + '/'

# Classifier 2: self-defined CNN model

In [19]:
```
#kares package use Tensorflow as backend
Xtr = Xtr.reshape(Xtr.shape[0], ydim, xdim, 1)
Xts = Xts.reshape(Xts.shape[0], ydim, xdim, 1)
test_rate = 0.2
Xtr, Xts, ytr, yts = train_test_split(X, y_raw, test_size=test_rate, r
andom_state=0)

#change the features to improve accurancy
rdim = Xtr.shape[0]
sdim = Xts.shape[0]
Xtr = Xtr.reshape(rdim, ydim, xdim)
Xts = Xts.reshape(sdim, ydim, xdim)

# feature selection method
# Xtr = np.delete(Xtr, [19,20], 1)
# Xts = np.delete(Xts, [19,20], 1)
print(Xtr.shape)
ydim = ydim - 0

# one hot encoded
ytr_reshape = ytr.reshape(-1, 1)
encoder = preprocessing.OneHotEncoder(sparse=False)
ytr_hot = encoder.fit_transform(ytr_reshape)
```

```
(2094, 56, 57)
```

In [20]:
```
## CNN model with 2 convolution layer

Xtr = Xtr.reshape(rdim, ydim, xdim, 1)
Xts = Xts.reshape(sdim, ydim, xdim, 1)

model = Sequential()

conv_filters = 32
in_shape = (ydim, xdim, 1)

# normalize for each batch
model.add(BatchNormalization(input_shape=in_shape))

# Layer 1
model.add(Convolution2D(conv_filters, (1,9), input_shape=in_shape))
model.add(MaxPooling2D(pool_size=(1,3)))
model.add(Dropout(0.1))

# Layer 2
model.add(Convolution2D(conv_filters, (1,9)))
model.add(MaxPooling2D(pool_size=(1,3)))
model.add(Dropout(0.1))

model.add(Flatten())

# model.add(Activation('relu'))
model.add(Dense(256, activation='sigmoid'))
model.add(Dropout(0.2))

# Output layer
model.add(Dense(7,activation='softmax'))
```

In [21]:
```
# ## CNN model with 2 parallel convolution layer

# Xtr = Xtr.reshape(rdim, ydim, xdim, 1)
# Xts = Xts.reshape(sdim, ydim, xdim, 1)

# conv_filters = 16
# in_shape = (ydim, xdim, 1)
# input = Input(in_shape)

# # parallel convolution layer on two dimensions
# conv_layer1 = Convolution2D(conv_filters, (9,1), activation='relu')(
input)  # a vertical filter
# conv_layer2 = Convolution2D(conv_filters, (1,9), activation='relu')(
input)  # a horizontal filter
# # conv_layer3 = Convolution1D(n_filters, (1,3), activation='relu')(i
nput[2])  # a horizontal filter

# # pooling layers
# maxpool1 = MaxPooling2D(pool_size=(1,3))(conv_layer1)
# maxpool2 = MaxPooling2D(pool_size=(3,1))(conv_layer2)

# # dropout layers
# maxpool1 = Dropout(0.1)(maxpool1)
# maxpool2 = Dropout(0.1)(maxpool2)

# # conv_layer3 = Convolution2D(n_filters, (9,1), activation='relu')(m
axpool1)
# # conv_layer4 = Convolution2D(n_filters, (1,9), activation='relu')(m
axpool2)
# # maxpool3 = MaxPooling2D(pool_size=(1,3))(conv_layer3)
# # maxpool4 = MaxPooling2D(pool_size=(3,1))(conv_layer4) # used 4,1 f
irst
# # maxpool3 = Dropout(0.25)(conv_layer3)
# # maxpool4 = Dropout(0.25)(conv_layer4)
# # maxpool4 = Dropout(0.25)(maxpool4)

# # flatten layers
# poolflat1 = Flatten()(maxpool1)
# poolflat2 = Flatten()(maxpool2)

# # Merge the 2 parallel pipelines
# merged = concatenate([poolflat1, poolflat2],1)

# full = Dense(256, activation='sigmoid')(merged)
# output_layer = Dense(7, activation='softmax')(full)

# # create the model
# model = Model(input=input, output=output_layer)
```

# Classifier 3: LSTM model

In [22]:
```
# model = Sequential()

# model.add(LSTM(512, return_sequences=True, input_shape=(ydim, xdim))
)
# model.add(Activation('relu'))
#   # model.add(LSTM(512, return_sequences=True))
#   # model.add(Activation('tanh'))
# model.add(LSTM(256, return_sequences=False))
# model.add(Activation('relu'))
# model.add(Dense(512, activation='sigmoid'))
# model.add(Dropout(0.5))
# model.add(Dense(7,activation='softmax'))
```

In [23]:
```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
batch_normalization_1 (Batch (None, 56, 57, 1)         4
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 56, 49, 32)        320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 56, 16, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 56, 16, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 56, 8, 32)         9248
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 56, 2, 32)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 56, 2, 32)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 3584)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               917760
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 7)                 1799
=================================================================
Total params: 929,131
Trainable params: 929,129
Non-trainable params: 2
_________________________________________________________________
```

In [24]:
```
# Define a loss function
loss = 'categorical_crossentropy'
# loss = 'sparse_categorical_crossentropy'

# use adam optimizer
opt = optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-0
8, decay=0.0)
model.compile(optimizer=opt,
              loss=loss,
              metrics=['accuracy'])
```

In [25]:
```
history = None
```

In [26]:
```
# train the model
# epochs numbers to train(common one)
nepochs = 10 # for self-build CNN
# nepochs = 20 # for LSTM only

# for training we need the "1 hot encoded" numeric classes of the grou
nd truth
validation_percent = 0.1
History = model.fit(Xtr, ytr_hot, validation_split=validation_percent,
batch_size=32, epochs=nepochs)

# keep history of accuracies on training set
# append to previous history in case multiple times are excuted
if history is None:
    history = History.history
else:
    for key in History.history.keys():
        history[key].extend(History.history[key])
```

```
Train on 1884 samples, validate on 210 samples
Epoch 1/10
1884/1884 [==============================] - 4s - loss: 1.0421 - acc
: 0.6274 - val_loss: 0.7942 - val_acc: 0.7190
Epoch 2/10
1884/1884 [==============================] - 4s - loss: 0.5504 - acc
: 0.8132 - val_loss: 0.6630 - val_acc: 0.7524
Epoch 3/10
1884/1884 [==============================] - 4s - loss: 0.3478 - acc
: 0.8737 - val_loss: 0.5511 - val_acc: 0.8143
Epoch 4/10
1884/1884 [==============================] - 4s - loss: 0.2170 - acc
: 0.9352 - val_loss: 0.5053 - val_acc: 0.8095
Epoch 5/10
1884/1884 [==============================] - 4s - loss: 0.1431 - acc
: 0.9613 - val_loss: 0.5006 - val_acc: 0.8333
Epoch 6/10
1884/1884 [==============================] - 4s - loss: 0.0913 - acc
: 0.9846 - val_loss: 0.4886 - val_acc: 0.8476
Epoch 7/10
1884/1884 [==============================] - 4s - loss: 0.0553 - acc
: 0.9936 - val_loss: 0.4967 - val_acc: 0.8238
Epoch 8/10
1884/1884 [==============================] - 4s - loss: 0.0373 - acc
: 0.9984 - val_loss: 0.4857 - val_acc: 0.8381
Epoch 9/10
1884/1884 [==============================] - 4s - loss: 0.0233 - acc
: 0.9989 - val_loss: 0.4819 - val_acc: 0.8476
Epoch 10/10
1884/1884 [==============================] - 4s - loss: 0.0169 - acc
: 1.0000 - val_loss: 0.4810 - val_acc: 0.8381
```

In [27]:
```
yts_pred = model.predict(Xts)
yts_pred = np.argmax(yts_pred, axis=1)
```

In [28]:
```
yts_pred.shape
```

Out[28]: (524,)

In [29]:
```
accuracy_score(yts, yts_pred)
```

Out[29]: 0.8492366412213741

In [30]:
```
size0 = np.shape(np.where(yts==0))[1]
size1 = np.shape(np.where(yts==1))[1]
size2 = np.shape(np.where(yts==2))[1]
size3 = np.shape(np.where(yts==3))[1]
size4 = np.shape(np.where(yts==4))[1]
size5 = np.shape(np.where(yts==5))[1]
size6 = np.shape(np.where(yts==6))[1]
size = np.array([1/size0, 1/size1, 1/size2, 1/size3, 1/size4, 1/size5,
1/size6])

dim = yts.shape[0]
# print(dim)
C = confusion_matrix(yts, yts_pred, labels=None, sample_weight=None)
# print(C)
C_normalized = size*C
print(np.array_str(C_normalized, precision=4))
```

```
[[ 0.9286  0.      0.      0.012   0.      0.0115  0.0405]
 [ 0.0143  0.9423  0.      0.      0.0256  0.      0.    ]
 [ 0.0143  0.      0.8125  0.0602  0.      0.046   0.0676]
 [ 0.      0.0769  0.025   0.8072  0.0128  0.0345  0.0811]
 [ 0.0429  0.1154  0.0375  0.0482  0.7564  0.023   0.0135]
 [ 0.0286  0.      0.0125  0.      0.      0.9655  0.    ]
 [ 0.0714  0.0385  0.025   0.012   0.0513  0.046   0.7568]]
```

In [ ]: