newliveclass.py

```
GLOBAL_TIME = 30

# This works similar to the classify.py script, but makes makes the process seamless
# by recording and classifying at the same time.

import platform
import os
import numpy as np
import pickle
import pyaudio
import wave
from scipy import signal
from librosa import load as lib_load
from librosa import feature as lib_feature
import librosa

# Import sci-kit models
from sklearn.preprocessing import OneHotEncoder, RobustScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import confusion_matrix
from sklearn.svm import SVC


def load_data(path, sample_rate, file):
    # Create data lists
    samples = None
    labels = None
    classes = os.listdir(path)
    # print('Loading data...')

    # for file in os.listdir(path):
    filename_path = os.path.join(path, file)
        # Load data
    samples, s = lib_load(filename_path, sr=sample_rate)
    # Append data and label
    # print('Loaded {}'.format(filename))

    return samples, classes


def get_subsamples(full_samples, num_samples=2, sub_time=2, bool=True):
    ## Multi-index sub-sampling
    # If True, data is down sampled. Specify subsampled time below.
    if bool:
        # Number of samples
        num_samples = num_samples
        # Specify down-sample time in seconds and number of samples
        subsample_time = sub_time
        subsample_samples = subsample_time * sample_rate
        # Create sub-sampled data
        subsamples = []
        # sublabels = []
        for idx, sample in enumerate(full_samples):
            # Get correct indicies to start
            start = 0
            end = start + subsample_samples
            for iidx in range(num_samples):
                if end >= len(sample):
                    break
                # Create sub-sampled array
                subsamples.append(sample[start:end])
                # Create new label array
                # sublabels.append(full_labels[idx])
                # Increment
                start = end + 1
                end = start + subsample_samples

        return subsamples
```

```python
# Extract MFCC features
def extract_mfcc(samples, sample_rate=44_100, num_mfcc=10, fft_size=1_024, window_numbers=20):
    # print('Extracting MFCCs...')
    features = []
    for i in range(len(samples)):
        # Compute MFCCs
        mfccs = lib_feature.mfcc(y=samples[i], sr=sample_rate, n_mfcc=10,
                                 win_length=int(np.ceil(fft_size / window_numbers)),
                                 hop_length=int(np.ceil(fft_size / (2 * window_numbers))))
        # Get mean for each value
        mfccs_mean = np.mean(mfccs.T, axis=0)
        # Re-cast to list
        mfccs_mean = mfccs_mean.tolist()

        # Get features for all samples
        features.append(mfccs_mean)

    # print('MFCCs extracted.')

    return features


# Extract Spectral Centorid features
def extract_sc(samples, sample_rate=44_100, fft_size=1_024, window_numbers=20):
    # print('Extracting spectral centroid...')
    features = []
    for i in range(len(samples)):
        # Compute Spectral Centroid
        sc = lib_feature.spectral_centroid(y=samples[i], sr=sample_rate,
                                           win_length=int(np.ceil(fft_size / window_numbers)),
                                           hop_length=int(np.ceil(fft_size / (2 * window_numbers))))
        reshape_sc = []
        for x in sc:
            for j in x:
                reshape_sc.append(j)
        max_sc = max(reshape_sc)
        min_sc = min(reshape_sc)
        mean_sc = np.mean(reshape_sc)
        sc_fv = [max_sc, min_sc, mean_sc]

        # Get features for all samples
        features.append(sc_fv)

    # print('Spectral Centroid extracted.')

    return features


# Extract Bandwidth features
def extract_bw(samples, sample_rate=44_100, fft_size=1_024, window_numbers=20):
    # print('Extracting bandwidth...')
    features = []
    for i in range(len(samples)):
        # Compute Bandwidth
        bw = lib_feature.spectral_bandwidth(y=samples[i], sr=sample_rate,
                                            win_length=int(np.ceil(fft_size / window_numbers)),
                                            hop_length=int(np.ceil(fft_size / (2 * window_numbers))))
        reshape_bw = []
        for x in bw:
            for j in x:
                reshape_bw.append(j)
        max_bw = max(reshape_bw)
        min_bw = min(reshape_bw)
        mean_bw = np.mean(reshape_bw)
        bw_fv = [max_bw, min_bw, mean_bw]

        # Get features for all samples
        features.append(bw_fv)

    # print('Bandwidth extracted.')

    return features
```

```python
def extract_features(samples):
    FFT_size = 1024

    # Define feature vector for classification
    features = []

    # Create initial feature vectors
    fvs = []
    # Enable features you wish to extract
    fvs.append(extract_mfcc(samples))
    fvs.append(extract_sc(samples))
    fvs.append(extract_bw(samples))

    # Loop over each data sample
    for idx, sample in enumerate(samples):
        # Concatenate all feature types for a sample
        sample_features = []
        for fv in fvs:
            sample_features.append(fv[idx])
        # Flatten list of features
        sample_features = sum(sample_features, [])

        # Append sample features to feature vector for classification
        features.append(sample_features)

    # print('Features extracted...')

    # Normalize feature
    scaler = RobustScaler()
    normalized_features = scaler.fit_transform(features)

    return normalized_features


def classify(clf, domain_fv):
    # print('Classifying...')
    # Specify which data to use, these are the only parameters that should change, the rest should remain the same.
    X = domain_fv

    # Convert X to numpy array if not imputing
    X = np.asarray(X)

    y_predict = clf.predict(X)

    return y_predict[0]


form_1 = pyaudio.paInt16 # 16-bit resolution
chans = 1 # 1 channel
samp_rate = 44100 # 44.1kHz sampling rate
chunk = 4096 # 2^12 samples for buffer
record_secs = GLOBAL_TIME # seconds to record
dev_index = 1 # device index found by p.get_device_info_by_index(ii)
wav_output_filename = 'live_recording.wav' # name of .wav file

audio = pyaudio.PyAudio() # create pyaudio instantiation

# create pyaudio stream
stream = audio.open(format=form_1, rate=samp_rate, channels=chans,
                    input_device_index=dev_index, input=True,
                    frames_per_buffer=chunk)
print("recording")

# Load model
pkl_filename = "pickle_model.pkl"

# print('Loading model...')
with open(pkl_filename, 'rb') as file:
    pickle_model = pickle.load(file)

class_history = np.full(10, fill_value=4)
```

```
#Record and classify loop
try:
    while True:


        print("recording in progress ... ")
        frames=[]   # reset frames array
        # loop through stream and append audio chunks to frame array
        for ii in range(0,int((samp_rate/chunk)*record_secs)):
            data = stream.read(chunk,exception_on_overflow=False)
            frames.append(data)

        print("finished recording")

        # save the audio frames as .wav file
        wavefile = wave.open(wav_output_filename,'wb')
        wavefile.setnchannels(chans)
        wavefile.setsampwidth(audio.get_sample_size(form_1))
        wavefile.setframerate(samp_rate)
        wavefile.writeframes(b''.join(frames))
        wavefile.close()

        # Get sample
        # Temporarily load data sample, replace with microphone
        audio_filename = 'live_recording.wav'
        path = os.getcwd()
        sample_rate = 44100
        samples, classes = load_data(path, sample_rate, audio_filename)
        classes = ['Dendropsophus bifurcus', 'Engystomops petersi', 'Pristimantis conspicillatus']

        subsamples = get_subsamples(np.array([samples]))

        # Feature Extraction
        features = extract_features(subsamples)

        # Classify
        pred = classify(pickle_model, features)
        np.roll(class_history,-1)
        class_history[-1] = pred

        # # classifier loop
        # if platform.system() == 'Windows':
        #     os.system('cls')
        # else:
        #     # for linux platfrom
        #     os.system('clear')

        print(classes[pred])
        # print("Predictions: \nnewest")
        hist_iter = 0
        # for prediction in class_history:
        #     live_label = "" if prediction == 4 else classes[prediction]
        #     print(f"\tprediction {hist_iter}: {live_label}")
        #     hist_iter += 1
            # print(f"Predicted: {classes[pred]}")
            # print(f"Expected: {label}")
        # print("oldest")

except KeyboardInterrupt:
    print("Stopping model")

# stop the stream, close it, and terminate the pyaudio instantiation
stream.stop_stream()
stream.close()
audio.terminate()
```