Musical Instrument Recognition in Solo-Instrument Recordings

MIR Course, April 2018

Final project for MIR course by Venkatesh Shenoy Kadandale, 2017-18 SMC Master Student

Objective

To classify sounds from Good-Sounds dataset based on the musical instrument category.

Methodology

Two approaches are presented. In the first approach, the models are trained and tested on the actual sounds from the dataset. Low overall accuracy is expected. In the second approach, the models are trained and tested over the sounds after they are stripped off all the sinusoidal components i.e the residual components of the sounds. The overall accuracy is expected to improve in the second approach. The silent frames are dropped and low-level statistical features are extracted using Essentia's MusicExtractor. Among these, the top five features in terms of the variance in distribution will be shortlisted. Support Vector Machines(SVM) are used for classification.

Dataset

Subset of Good-Sounds Dataset. Here's the link to the complete dataset. The original dataset is provided with a CC BY-NC 4.0 license. I have used only a subset of this dataset for this task. The original dataset has the following folder structure:

```
.
 |-sax_alto_scale_raul_recordings
 |   |-neumann
 |   |-iphone
 |-saxo_tenor_raul_recordings
 |   |-neumann
 |-trumpet_ramon_timbre_stability
 |   |-neumann
 |   |-akg
 |-saxo_soprane_raul_recordings
 |   |-neumann
 |   |-iphone
 |-flute_almudena_evaluation_recordings
 |   |-iphone
 |-clarinet_pablo_attack
 |   |-neumann
 |   |-akg
 |-clarinet_pablo_timbre_stability
```

```
|  |-neumann
|  |-akg
|-cello_nico_improvement_recordings
|  |-neumann
|  |-iphone
|-piccolo_irene_recordings
|  |-neumann
|  |-iphone
|-trumpet_jesus_improvement_recordings
|  |-neumann
|  |-iphone
|-trumpet_ramon_attack_stability
|  |-neumann
|  |-akg
|-violin_raquel_attack
|  |-neumann
|  |-akg
|-saxo_raul_recordings
|  |-neumann
|  |-iphone
|-clarinet_gener_evaluation_recordings
|  |-iphone
|-bass_alejandro_recordings
|  |-neumann
|-violin_laia_improvement_recordings
|  |-neumann
|-cello_margarita_open_strings
|  |-neumann
|  |-akg
|  |-iphone
|-flute_almudena_reference
|  |-neumann
|  |-akg
|  |-iphone
|-violin_violin_scales_laia_recordings
|  |-neumann
|-flute_almudena_attack
|  |-neumann
|  |-akg
|  |-iphone
|-oboe_marta_recordings
|  |-neumann
|  |-iphone
|-clarinet_pablo_dynamics_stability
|  |-neumann
|  |-akg
|-flute_almudena_air
|  |-neumann
|  |-akg
|  |-iphone
|-trumpet_ramon_reference
```

```
|   |-neumann
|   |-akg
|   |-iphone
|-clarinet_marti_evaluation_recordings
|   |-iphone
|-trumpet_ramon_dynamics_stability
|   |-neumann
|   |-akg
|-flute_almudena_reference_piano
|   |-neumann
|   |-akg
|   |-iphone
|-trumpet_jesus_evaluation_recordings
|   |-iphone
|-cello_margarita_attack
|   |-neumann
|   |-akg
|-sax_alto_scale_2_raul_recordings
|   |-neumann
|   |-iphone
|-cello_margarita_reference
|   |-neumann
|   |-akg
|-flute_josep_evaluation_recordings
|   |-iphone
|-violin_raquel_richness
|   |-neumann
|   |-akg
|-flute_scale_irene_recordings
|   |-neumann
|   |-iphone
|-sax_tenor_tenor_scales_2_raul_recordings
|   |-neumann
|-flute_almudena_stability
|   |-neumann
|   |-akg
|   |-iphone
|-clarinet_pablo_air
|   |-neumann
|   |-akg
|-clarinet_gener_improvement_recordings
|   |-neumann
|   |-iphone
|-violin_raquel_reference
|   |-neumann
|   |-akg
|-clarinet_scale_gener_recordings
|   |-neumann
|   |-iphone
|-saxo_bariton_raul_recordings
|   |-neumann
```

```
|   |-iphone
|-cello_margarita_dynamics_stability
|   |-neumann
|   |-akg
|-cello_margarita_timbre_stability
|   |-neumann
|   |-akg
|-flute_josep_improvement_recordings
|   |-neumann
|   |-iphone
|-trumpet_ramon_evaluation_recordings
|   |-iphone
|-cello_margarita_timbre_richness
|   |-neumann
|   |-akg
|-violin_raquel_timbre_stability
|   |-neumann
|   |-akg
|-trumpet_ramon_air
|   |-neumann
|   |-akg
|-sax_tenor_tenor_scales_raul_recordings
|   |-neumann
|-clarinet_pablo_richness
|   |-neumann
|   |-akg
|-violin_raquel_dynamics_stability
|   |-neumann
|   |-akg
|-flute_almudena_dynamics_change
|   |-neumann
|   |-akg
|   |-iphone
|-clarinet_pablo_pitch_stability
|   |-neumann
|   |-akg
|-trumpet_scale_jesus_recordings
|   |-neumann
|   |-iphone
|-flute_almudena_timbre
|   |-neumann
|   |-akg
|   |-iphone
|-trumpet_ramon_pitch_stability
|   |-neumann
|   |-akg
|-saxo_tenor_iphone_raul_recordings
|   |-iphone
|-violin_raquel_pitch_stability
|   |-neumann
|   |-akg
```

```
|-cello_margarita_pitch_stability
|   |-neumann
|   |-akg
|-clarinet_pablo_reference
|   |-neumann
|   |-akg
|-violin_laia_improvement_recordings_2
|   |-iphone
```

All the audio clips are segregated into folders based on the instrument categories: bass, cello, clarinet, flute, oboe, piccolo, saxophone, trumpet and violin. The following is the resulting folder structure:

```
.
|-sax(3360 samples)
|-pic(776 samples)
|-flu(2308 samples)
|-vio(1853 samples)
|-cel(2118 samples)
|-obo(494 samples)
|-cla(3359 samples)
|-bas(159 samples)
|-tru(1883 samples)
```

As we see, dataset is severely unbalanced. To have a uniform distribution of data among all the instrument categories, I have randomly chosen 159 samples from each of the categories. This pruned and restructured version of the original dataset is considered for this project. The sounds from this subset are further split into their sinusoidal and residual components. The Essentia music extractor has been used to extract the lowlevel statistical features for all these three categories: original sound, residual and sinusoidal. All these processed/pre-extracted data are temporarily made available here.

NOTE:

The dataset, by itself, is not best suited for instrument classification task as the sound categories being considered, do not represent the diversity of musical instruments. There are no percussion instruments. The sound samples are categorized not just based on instruments but also by the way they are played. These sub-categorizations give an in-depth representation of the instrument. It makes sense to treat sub-categories as different instruments since their textures are drastically different in some cases. For example, a violin pluck sounds totally different from a bowed sound. Also, some sounds involve noise generated by the musician. For example, breathing sounds while playing flute. Again, these sounds are not there in all the flute samples, they have critical presence in some sub-categories. Due to all these factors, we need to decide which instrument sub-categories are to be merged and which ones need to be considered seperate. This involves careful inspection of sounds from all the 61 sub-categories. This is beyond the scope of this project. In this project, we will be merging all the sounds based on instruments without taking into account the sub-category(which tells us how it was played). Also, sax-alto, sax-tenor, sax-soprano and sax-baritone are merged. One more drawback is that the dataset is unbalanced, which forces us to remove samples from the dataset so that each class has same number of samples. Also, for this project, the sounds are randomly selected. Hence, the distribution of sounds with respect to sub-categories of sounds are not uniform. The project does not aim to arrive at the feature set or the classifier parameters that gives the best classification accuracy. The objective here is to study the effect of dataset refinement(separating out the residual/sinusoidal) on classification accuracy keeping the feature set and classifier parameters fixed.

In [1]:

```
# all the imports
import io,math
import os,sys
import urllib
import zipfile
import json
import itertools
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import display, HTML, Audio, display_html

CSS = """
.output {
    flex-direction: row;
}
"""

HTML('<style>{}</style>'.format(CSS))


import essentia
import essentia.standard as es
import pandas as pd #python library for data manipulation and analysis
import seaborn as sns; # for visualizing data

from sklearn import svm #libraries for machine learning
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.feature_selection import SelectFromModel
from sklearn.svm import LinearSVC
from sklearn.feature_selection import SelectKBest

#external .py files
import download_file_from_google_drive #for downloading big files from goog
le drive
import confirm_prompt #for confirming user action
import json_flattener #for flatenning jsons
```

Downloading the pre-compiled dataset.

As discussed in **Dataset** section above, we need to get the original sounds, residuals and the sinusoids. The next code segments will provide the user an option to download them or skip downloading if the user already has them(all the three folders!).

In [2]:

```
#This is where the path of dataset is set. Within 'instruments' folder, we
need to have sub-folders {bas, cel ...}
#The residuals and sinusoids data will follow the same hierarchy as the
'instruments' folder.
path_to_dataset='../../data/good-sounds/instruments/'
```

```
path_to_dataset='../../data/good-sounds/instruments/'
path_to_sinusoids='../../data/good-sounds/sinusoids/'
path_to_residuals='../../data/good-sounds/residuals/'
```

In [3]:

```
if not os.path.exists(path_to_dataset):
    os.umask(0) #To mask the permission restrictions on new
files/directories being created
    os.makedirs(path_to_dataset,0o777) # 0o777 gives us full permissions fo
r the folder

if not os.path.exists(path_to_sinusoids):
    os.umask(0) #To mask the permission restrictions on new
files/directories being created
    os.makedirs(path_to_sinusoids,0o777) # 0o777 gives us full permissions
for the folder

if not os.path.exists(path_to_residuals):
    os.umask(0) #To mask the permission restrictions on new
files/directories being created
    os.makedirs(path_to_residuals,0o777) # 0o777 gives us full permissions
for the folder


# Prompt to know if you want to skip downloading the dataset
skip_dataset_download=confirm_prompt.confirm(prompt='Would you like to skip
downloading the data? \nEnter \'y\' if you already have dataset and \'n\' t
o download the dataset.\n [NOTE] : Downloading the dataset using this
notebook can take up to half an hour.\n')
if(not skip_dataset_download):

    #This block downloads the originals from google drive
    file_id='1Nj4SsjCwwwEzYkYatLHYjWXOOGTT_wv3'
    filename=path_to_dataset+"instruments.zip"
    print("Downloading the originals dataset...")
    download_file_from_google_drive.download_file_from_google_drive(file_id
,filename)
    print("Unzipping the data file.")
    #Unzip the file
    zip_ref = zipfile.ZipFile(filename, 'r')
    zip_ref.extractall(path_to_dataset)
    zip_ref.close()
    os.remove(filename)#Removing the zip file
    print('Originals downloaded and unzipped to: ',path_to_dataset)

#---------------------------------------------------------------------------
--------------------------------

    #This block downloads the sinusoids from google drive
    file_id='17Br8mLdFkmtpOwW34-07kOUkBxGY5kzN'
    filename=path_to_sinusoids+"sinusoids.zip"
    print("Downloading the sinusoids dataset...")
    download_file_from_google_drive.download_file_from_google_drive(file_id
,filename)
    print("Unzipping the data file.")
    #Unzip the file
    zip_ref = zipfile.ZipFile(filename, 'r')
    zip_ref.extractall(path_to_sinusoids)
    zip_ref.close()
    os.remove(filename)#Removing the zip file
```

```
    os.remove(filename)#Removing the zip file
    print('Sinusoids downloaded and unzipped to: ',path_to_sinusoids)

#------------------------------------------------------------------------------------------------

    #This block downloads the residuals from google drive
    file_id='1j1l87bs_fo4XvYpsITglABGGQ_GB-YCD'
    filename=path_to_residuals+"residuals.zip"
    print("Downloading the residuals dataset...")
    download_file_from_google_drive.download_file_from_google_drive(file_id
,filename)
    print("Unzipping the data file.")
    #Unzip the file
    zip_ref = zipfile.ZipFile(filename, 'r')
    zip_ref.extractall(path_to_residuals)
    zip_ref.close()
    os.remove(filename)#Removing the zip file
    print('Residuals downloaded and unzipped to: ',path_to_residuals)

    print('The datasets have been successfully
downloaded',path_to_residuals)

# [NOTE] If you already have the dataset, move the instrument folders {bas,
cel ...} into path_to_dataset
instruments = os.listdir(path_to_dataset)
instruments.sort()
```

```
Would you like to skip downloading the data?
Enter 'y' if you already have dataset and 'n' to download the dataset.
 [NOTE] : Downloading the dataset using this notebook can take up to half a
n hour.
y
```

NOTE: The residual and sinusoid datasets are built by splitting each of the sounds into residual and sinusoidal components. The scripts sinusoidalSeparator .py and residualSeparator.py (included in the submission) are used to extract these components. The scripts internally make use of sms-tools models sprModel to separate out the residual and sinusoidal components.

In [4]:

```
from IPython.core.display import HTML

def multi_table(table_list):
    ''' Acceps a list of IpyTable objects and returns a table which contain
s each IpyTable in a cell
    '''
    return HTML(
        '<table><tr style="background-color:white;">' +
        ''.join(['<td>' + table._repr_html_() + '</td>' for table in
table_list]) +
        '</tr></table>'
    )
```

In [5]:

```
# A sample from Clarinet
import IPython
print(" [ORIGINAL]                    =        [SINUSOID]             +
[RESIDUAL]")
multi_table(

[Audio(os.path.join(path_to_dataset,'cla/clarinet_pablo_dynamics_stability_
umann_0117.wav')),

Audio(os.path.join(path_to_sinusoids,'cla/clarinet_pablo_dynamics_stability
eumann_0117_sinusoid.wav')),

Audio(os.path.join(path_to_residuals,'cla/clarinet_pablo_dynamics_stability
eumann_0117_residual.wav'))]
)
```

```
 [ORIGINAL]                    =        [SINUSOID]             +         [RE
IDUAL]
```

Out[5]:

Your browser does not support the audio element.	Your browser does not support the audio element.	Your browser does not support the audio element.

Feature Extraction

We extract all the low level features using Essentia's Music Extractor. This needs to be done offline using the script featureExtractor.py. We have already done this and we will be downloading the pre-extracted jsons to save time.

In [6]:

```
###### Prompt to know if you want to download pre-extracted features jsons
or you already have them
skip_json_download=confirm_prompt.confirm(prompt='Would you like to skip do
wnloading the feature jsons(Opt for skipping only if you already have the
feature jsons)? \nEnter \'y\' if you want to skip re-downloading and \'n\'
to download pre-extracted jsons.\n')
if(not skip_json_download):

    #Download the pre-extracted feature jsons of Original sounds
    file_id='145dZKM1kckq1dY2lTBs0aFx5oLkL1YGC'
    filename=path_to_dataset+"instrument_jsons.zip"
    print("Downloading the pre-extracted jsons for [Originals]...")

    #For Python2
    #urllib.urlretrieve('http://docs.google.com/uc?id='+file_id,filename)

    #For Python3
    urllib.request.urlretrieve('http://docs.google.com/uc?id='+file_id,file
name)

    #Unzip the file
    zip_ref = zipfile.ZipFile(filename, 'r')
    zip_ref.extractall(path_to_dataset)
```

```
    zip_ref.extractall(path_to_dataset)
    zip_ref.close()
    os.remove(filename)#Removing the zip file
    print('Jsons for [ORIGINAL] sounds have been downloaded and unzipped to the instrument specific folders.')

    #-------------------------------------------------------------------------------------------------------------

    #Download the pre-extracted feature jsons of Sinusoid sounds
    file_id='14_xbosKTkFdzYSg3_Hm5MPjxmKwBt-ny'
    filename=path_to_sinusoids+"sinusoid_instrument_jsons.zip"
    print("Downloading the pre-extracted jsons for [SINUSOIDS]...")

    #For Python2
    #urllib.urlretrieve('http://docs.google.com/uc?id='+file_id,filename)

    #For Python3
    urllib.request.urlretrieve('http://docs.google.com/uc?id='+file_id,filename)

    #Unzip the file
    zip_ref = zipfile.ZipFile(filename, 'r')
    zip_ref.extractall(path_to_sinusoids)
    zip_ref.close()
    os.remove(filename)#Removing the zip file
    print('Jsons for [SINUSOID] sounds have been downloaded and unzipped to the instrument specific folders.')

    #-------------------------------------------------------------------------------------------------------------

    #Download the pre-extracted feature jsons of Original sounds
    file_id='1k5NBG2TkIfFNnY6uU9CgWEibdV63oSEB'
    filename=path_to_residuals+"residual_instrument_jsons.zip"
    print("Downloading the pre-extracted jsons for [RESIDUALS]...")

    #For Python2
    #urllib.urlretrieve('http://docs.google.com/uc?id='+file_id,filename)

    #For Python3
    urllib.request.urlretrieve('http://docs.google.com/uc?id='+file_id,filename)

    #Unzip the file
    zip_ref = zipfile.ZipFile(filename, 'r')
    zip_ref.extractall(path_to_residuals)
    zip_ref.close()
    os.remove(filename)#Removing the zip file
    print('Jsons for [RESIDUAL] sounds have been downloaded and unzipped to the instrument specific folders.')

    print("JSONs have been successfully extracted")
```

```
Would you like to skip downloading the feature jsons(Opt for skipping only if you already have the feature jsons)?
Enter 'y' if you want to skip re-downloading and 'n' to download pre-extracted jsons.
y
```

NOTE: The features have been extracted externally using the python script featureExtractor.py included in the submission. This internally makes use of Essentia's MusicExtractor to extract the features.

Feature Selection

We flatten the json and choose the features that we are interested in: ['lowlevel_hfc_mean', 'lowlevel_dissonance_mean', 'lowlevel_pitch_salience_mean', 'lowlevel_spectral_flux_mean', 'lowlevel_zerocrossingrate_mean', 'lowlevel_hfc_stdev', 'lowlevel_dissonance_stdev', 'lowlevel_pitch_salience_stdev', 'lowlevel_spectral_flux_stdev', 'lowlevel_zerocrossingrate_stdev]

In [7]:

```
sortedFeatures=sorted(['lowlevel_hfc_mean',
                       'lowlevel_dissonance_mean',
                       'lowlevel_pitch_salience_mean',
                       'lowlevel_spectral_flux_mean',
                       'lowlevel_zerocrossingrate_mean',
                       'lowlevel_hfc_stdev',
                       'lowlevel_dissonance_stdev',
                       'lowlevel_pitch_salience_stdev',
                       'lowlevel_spectral_flux_stdev',
                       'lowlevel_zerocrossingrate_stdev',
                      ])
features=['filename','instrument']
features.extend(sortedFeatures)

# [ORIGINAL] Load data into Pandas Dataframes
dictValues={}
dfv_original=pd.DataFrame(dictValues, columns=features)
i=0
for instrument in instruments:
    print("Fetching json files from [INSTRUMENT] : " + instrument)
    files=os.listdir(path_to_dataset+instrument)
    for fileName in files:
        if(fileName.endswith('.json')):
            jsonFile = io.open(path_to_dataset+instrument+"/"+fileName,"r",e
ncoding="utf-8")
            jsonToPython = json.loads(jsonFile.read(), strict=False)
            flatJson = json_flattener.flatten_json(jsonToPython)
            dictValues[features[0]] = fileName.replace('.json','')
            dictValues[features[1]] = instruments.index(instrument)+1
            for index in range(2,len(features)):
                dictValues[features[index]]=flatJson.get(features[index])
            dfv_original.loc[i]=(dictValues)
            i+=1
print("Features loaded into panda dataframe!")

#---------------------------------------------------------------------------
--------------------------------

# [SINUSOIDAL] Load data into Pandas Dataframes
dictValues={}
dfv_sinusoidal=pd.DataFrame(dictValues, columns=features)
i=0
for instrument in instruments:
    print("Fetching json files from [SINUSOIDS] : " + instrument)
```

```
    print("Fetching json files from [SINUSOIDS] : " + instrument)
    files=os.listdir(path_to_sinusoids+instrument)
    for fileName in files:
        if(fileName.endswith('.json')):
            jsonFile = io.open(path_to_sinusoids+instrument+"/"+fileName,"r'
,encoding="utf-8")
            jsonToPython = json.loads(jsonFile.read(), strict=False)
            flatJson = json_flattener.flatten_json(jsonToPython)
            dictValues[features[0]] = fileName.replace('.json','')
            dictValues[features[1]] = instruments.index(instrument)+1
            for index in range(2,len(features)):
                dictValues[features[index]]=flatJson.get(features[index])
            dfv_sinusoidal.loc[i]=(dictValues)
            i+=1
print("[SINUSOID] Features loaded into panda [SINUSOID] dataframe!")

#---------------------------------------------------------------------------
--------------------------------

# [RESIDUAL] Load data into Pandas Dataframes
dictValues={}
dfv_residual=pd.DataFrame(dictValues, columns=features)
i=0
for instrument in instruments:
    print("Fetching json files from [RESIDUALS] : " + instrument)
    files=os.listdir(path_to_residuals+instrument)
    for fileName in files:
        if(fileName.endswith('.json')):
            jsonFile = io.open(path_to_residuals+instrument+"/"+fileName,"r'
,encoding="utf-8")
            jsonToPython = json.loads(jsonFile.read(), strict=False)
            flatJson = json_flattener.flatten_json(jsonToPython)
            dictValues[features[0]] = fileName.replace('.json','')
            dictValues[features[1]] = instruments.index(instrument)+1
            for index in range(2,len(features)):
                dictValues[features[index]]=flatJson.get(features[index])
            dfv_residual.loc[i]=(dictValues)
            i+=1
print("[RESIDUAL] Features loaded into panda [RESIDUAL] dataframe!")
```

```
Fetching json files from [INSTRUMENT] : bas
Fetching json files from [INSTRUMENT] : cel
Fetching json files from [INSTRUMENT] : cla
Fetching json files from [INSTRUMENT] : flu
Fetching json files from [INSTRUMENT] : obo
Fetching json files from [INSTRUMENT] : pic
Fetching json files from [INSTRUMENT] : sax
Fetching json files from [INSTRUMENT] : tru
Fetching json files from [INSTRUMENT] : vio
Features loaded into panda dataframe!
Fetching json files from [SINUSOIDS] : bas
Fetching json files from [SINUSOIDS] : cel
Fetching json files from [SINUSOIDS] : cla
Fetching json files from [SINUSOIDS] : flu
Fetching json files from [SINUSOIDS] : obo
Fetching json files from [SINUSOIDS] : pic
Fetching json files from [SINUSOIDS] : sax
Fetching json files from [SINUSOIDS] : tru
Fetching json files from [SINUSOIDS] : vio
[SINUSOID] Features loaded into panda [SINUSOID] dataframe!
```

```
Fetching json files from [RESIDUALS] : bas
Fetching json files from [RESIDUALS] : cel
Fetching json files from [RESIDUALS] : cla
Fetching json files from [RESIDUALS] : flu
Fetching json files from [RESIDUALS] : obo
Fetching json files from [RESIDUALS] : pic
Fetching json files from [RESIDUALS] : sax
Fetching json files from [RESIDUALS] : tru
Fetching json files from [RESIDUALS] : vio
[RESIDUAL] Features loaded into panda [RESIDUAL] dataframe!
```

Standardize features to zero mean and unit variance

In [8]:

```
scaler = StandardScaler() #To standardize the features to zero mean and uni
t variance

df1_original = dfv_original.iloc[:, :2] #seperate out filenames and instrum
ent columns from the rest
df2_original = dfv_original.iloc[:, 2:]
df2_original[df2_original.columns] = scaler.fit_transform(df2_original[df2_
original.columns])

#---------------------------------------------------------------------------
--------------------------------

df1_sinusoidal = dfv_sinusoidal.iloc[:, :2] #seperate out filenames and ins
trument columns from the rest
df2_sinusoidal = dfv_sinusoidal.iloc[:, 2:]
df2_sinusoidal[df2_sinusoidal.columns] =
scaler.fit_transform(df2_sinusoidal[df2_sinusoidal.columns])

#---------------------------------------------------------------------------
--------------------------------

df1_residual = dfv_residual.iloc[:, :2] #seperate out filenames and instrum
ent columns from the rest
df2_residual = dfv_residual.iloc[:, 2:]
df2_residual[df2_residual.columns] = scaler.fit_transform(df2_residual[df2_
residual.columns])
```

Segregating Train and Test data

In [9]:

```
#Create train_df for training data(80% of dataset) and test_df for test da
ta(20% of dataset).
#if dataset is balanced across all the classes
class_size=int(len(df1_original)/len(instruments))
train_size=int(math.floor(.8*class_size))
test_size=int(math.ceil(.2*class_size))
index=np.arange(10)*class_size


df_original = pd.concat([df1_original, df2_original], axis=1)
```

```
df_sinusoidal = pd.concat([df1_sinusoidal, df2_sinusoidal], axis=1)
df_residual = pd.concat([df1_residual, df2_residual], axis=1)

train_df_original=pd.DataFrame({}, columns=features)
train_df_sinusoidal=pd.DataFrame({}, columns=features)
train_df_residual=pd.DataFrame({}, columns=features)

for i in range(len(instruments)):
    train_df_original = pd.concat([train_df_original,df_original.iloc[index
[i]:index[i]+train_size,:]], ignore_index=True)
    train_df_sinusoidal = pd.concat([train_df_sinusoidal,df_sinusoidal.iloc
[index[i]:index[i]+train_size,:]], ignore_index=True)
    train_df_residual = pd.concat([train_df_residual,df_residual.iloc[index
[i]:index[i]+train_size,:]], ignore_index=True)

X_original = train_df_original.iloc[:,2:].as_matrix()
y_original = train_df_original.transpose().as_matrix()[1].astype('int')

X_sinusoidal = train_df_sinusoidal.iloc[:,2:].as_matrix()
y_sinusoidal = train_df_sinusoidal.transpose().as_matrix()[1].astype('int')

X_residual = train_df_residual.iloc[:,2:].as_matrix()
y_residual = train_df_residual.transpose().as_matrix()[1].astype('int')


test_df_original=pd.DataFrame({}, columns=features)
test_df_sinusoidal=pd.DataFrame({}, columns=features)
test_df_residual=pd.DataFrame({}, columns=features)
for i in range(len(instruments)):
    test_df_original = pd.concat([test_df_original,df_original.iloc[index[i
]+train_size:index[i+1],:]], ignore_index=True)
    test_df_sinusoidal = pd.concat([test_df_sinusoidal,df_sinusoidal.iloc[i
ndex[i]+train_size:index[i+1],:]], ignore_index=True)
    test_df_residual = pd.concat([test_df_residual,df_residual.iloc[index[i
]+train_size:index[i+1],:]], ignore_index=True)
```

Training

In [10]:

```
clf_original = LinearSVC()
clf_original.fit(X_original, y_original)
clf_sinusoidal = LinearSVC()
clf_sinusoidal.fit(X_sinusoidal, y_sinusoidal)
clf_residual = LinearSVC()
clf_residual.fit(X_residual, y_residual)
```

Out[10]:

```
LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
     intercept_scaling=1, loss='squared_hinge', max_iter=1000,
     multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
     verbose=0)
```

Testing

In [11]:

```
test_data_original=test_df_original.iloc[:,2:].as_matrix()
clf_output_original = clf_original.predict(test_data_original) # storing cl
assifier output - predicted labels
gt_original = test_df_original.transpose().as_matrix()[1].astype('int') # s
toring ground truth

test_data_sinusoidal=test_df_sinusoidal.iloc[:,2:].as_matrix()
clf_output_sinusoidal = clf_sinusoidal.predict(test_data_sinusoidal) # stor
ing classifier output - predicted labels
gt_sinusoidal = test_df_sinusoidal.transpose().as_matrix()[1].astype('int')
# storing ground truth

test_data_residual=test_df_residual.iloc[:,2:].as_matrix()
clf_output_residual = clf_residual.predict(test_data_residual) # storing cl
assifier output - predicted labels
gt_residual = test_df_residual.transpose().as_matrix()[1].astype('int') # s
toring ground truth
```

Evaluation

In [12]:

```
# Compute confusion matrix
cnf_matrix_original = confusion_matrix(gt_original, clf_output_original)
cnf_matrix_sinusoidal = confusion_matrix(gt_sinusoidal,
clf_output_sinusoidal)
cnf_matrix_residual = confusion_matrix(gt_residual, clf_output_residual)

np.set_printoptions(precision=2)
class_names=instruments
```

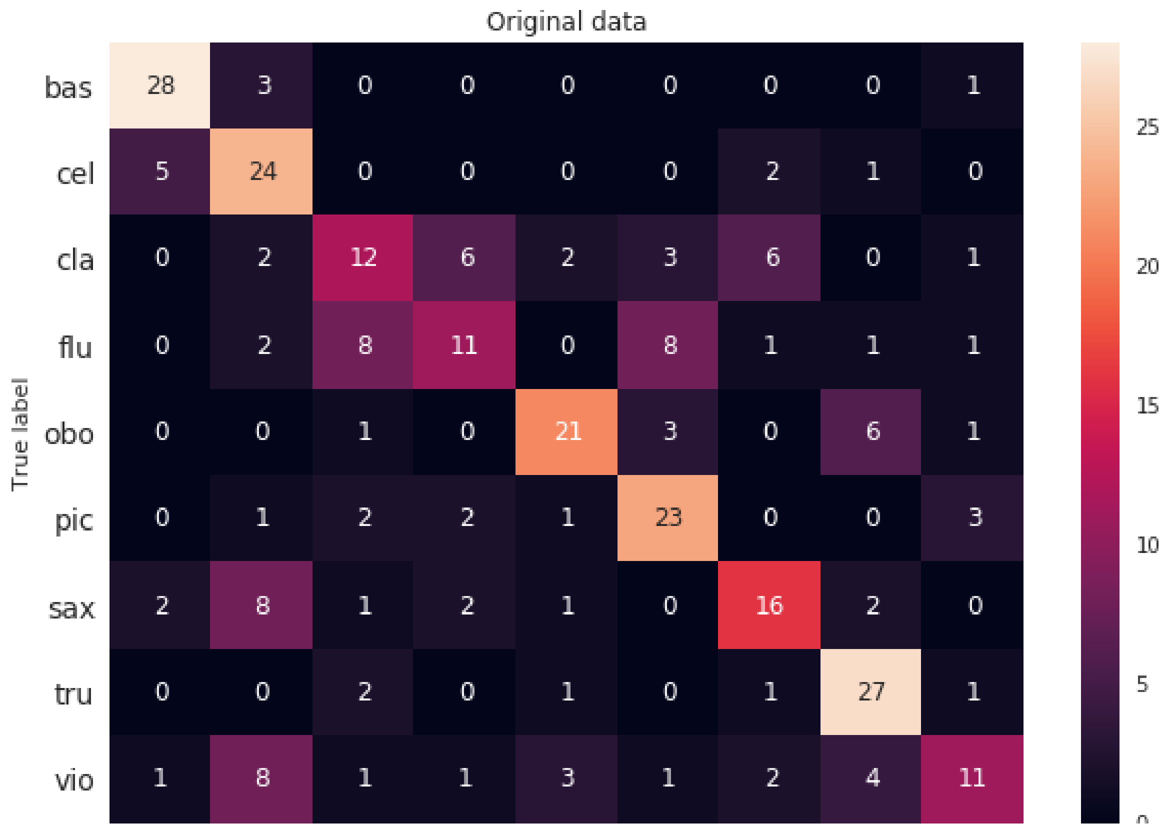
Results

In [14]:

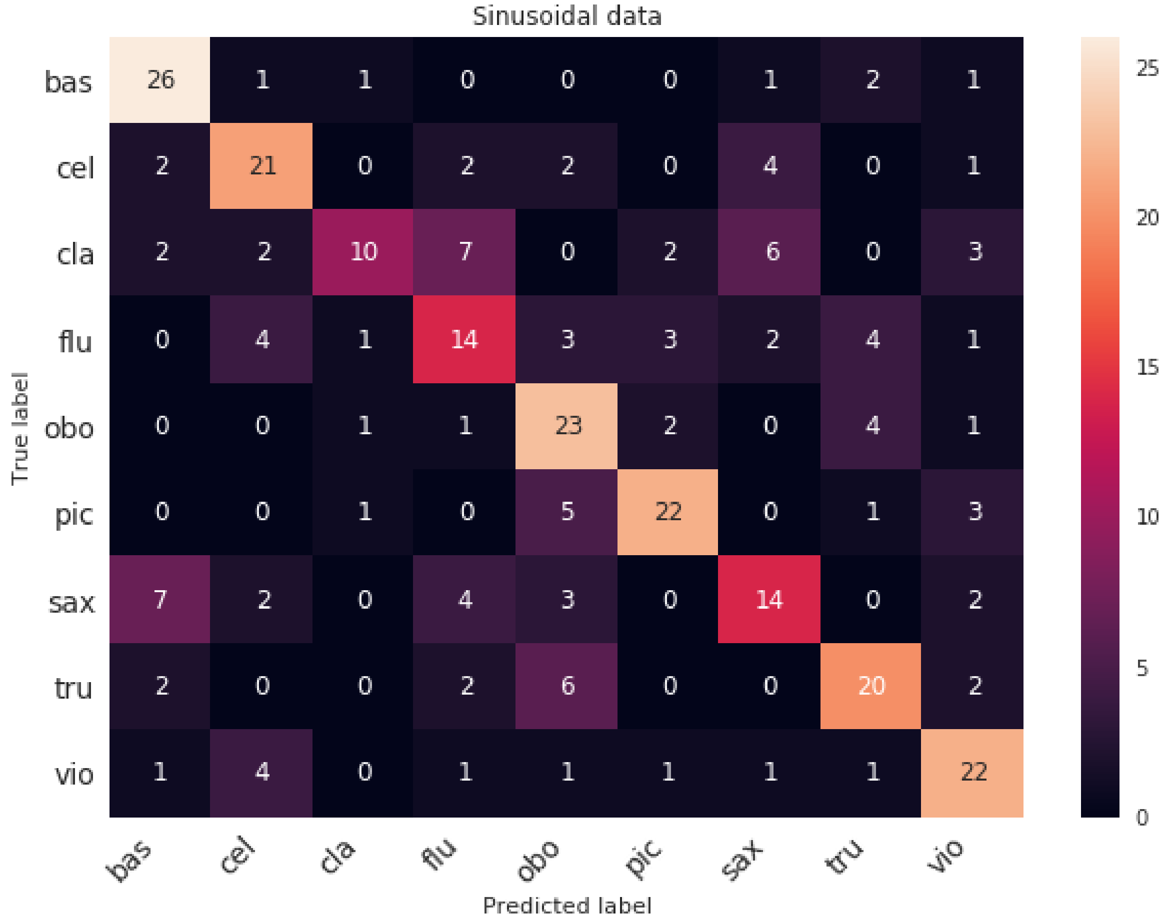
```
# A seaborn heatmap is used to visualize the confusion matrix
sns.set()
df_cm = pd.DataFrame(cnf_matrix_original, index=class_names, columns=class_
names)
fig = plt.figure(figsize=(10,7))
heatmap = sns.heatmap(df_cm, annot=True, fmt="d")
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha
='right', fontsize=14)
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=45, h
a='right', fontsize=14)
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.title('Original data')
print("Classification accuracy with [ORIGINALS]: "+str(100*accuracy_score(g
t_original,clf_output_original))+" %")

#---------------------------------------------------------------------------
--------------------------------

# A seaborn heatmap is used to visualize the confusion matrix
sns.set()
df_cm = pd.DataFrame(cnf_matrix_sinusoidal, index=class_names, columns=clas
```

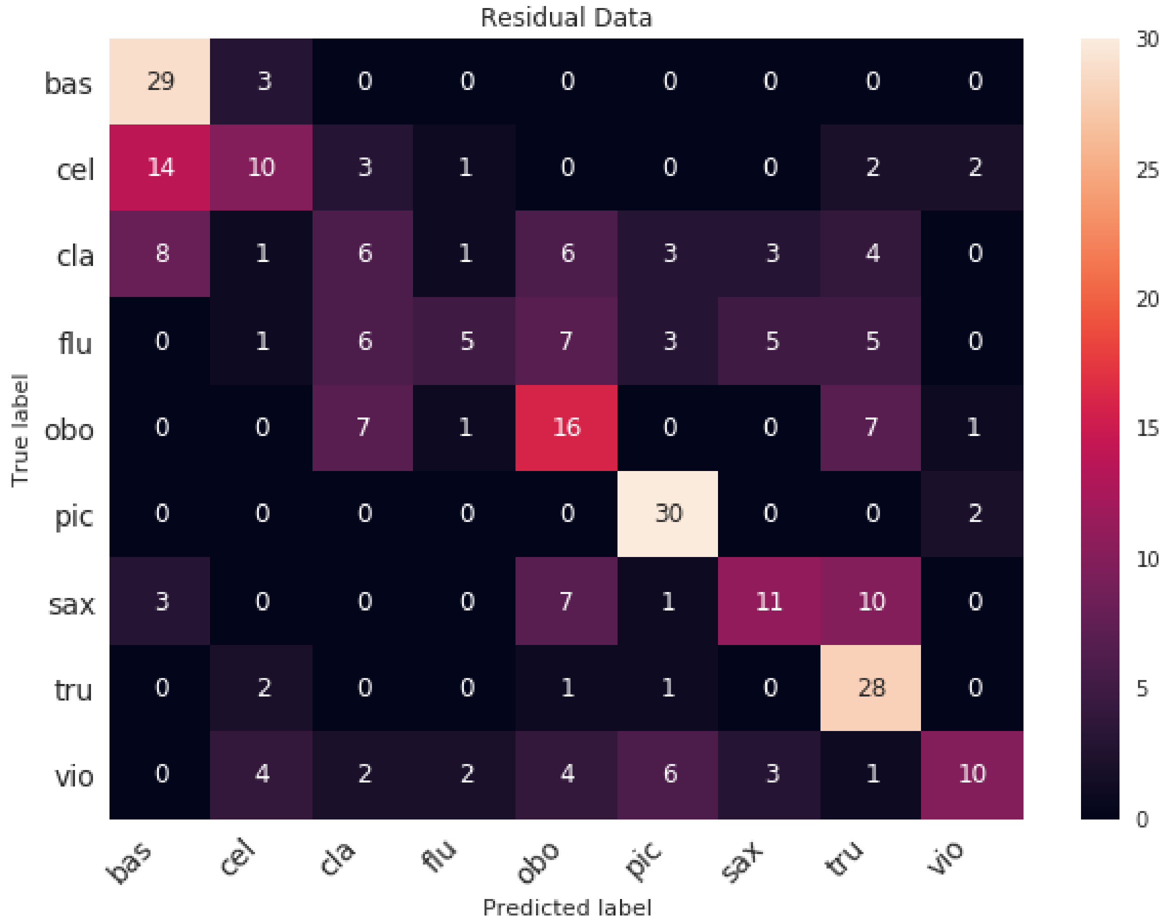
```
s_names)
fig = plt.figure(figsize=(10,7))
heatmap = sns.heatmap(df_cm, annot=True, fmt="d")
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha
='right', fontsize=14)
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=45, h
a='right', fontsize=14)
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.title('Sinusoidal data')
print("Classification accuracy with [SINUSOIDALS] : "+str(100*accuracy_scor
e(gt_sinusoidal,clf_output_sinusoidal))+" %")

#---------------------------------------------------------------------------
--------------------------------

# A seaborn heatmap is used to visualize the confusion matrix
sns.set()
df_cm = pd.DataFrame(cnf_matrix_residual, index=class_names, columns=class_
names)
fig = plt.figure(figsize=(10,7))
heatmap = sns.heatmap(df_cm, annot=True, fmt="d")
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha
='right', fontsize=14)
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=45, h
a='right', fontsize=14)
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.title('Residual Data')
print("Classification accuracy with [RESIDUALS] : "+str(100*accuracy_score(
gt_residual,clf_output_residual))+" %")
```

```
Classification accuracy with [ORIGINALS]: 60.0694444444 %
Classification accuracy with [SINUSOIDALS] : 59.7222222222 %
Classification accuracy with [RESIDUALS] : 50.3472222222 %
```




bas cel cla flu obo pic sax tru vio
Predicted label
0


Sinusoidal data
True label
bas 26 1 1 0 0 0 1 2 1
cel 2 21 0 2 2 0 4 0 1
cla 2 2 10 7 0 2 6 0 3
flu 0 4 1 14 3 3 2 4 1
obo 0 0 1 1 23 2 0 4 1
pic 0 0 1 0 5 22 0 1 3
sax 7 2 0 4 3 0 14 0 2
tru 2 0 0 2 6 0 0 20 2
vio 1 4 0 1 1 1 1 1 22
bas cel cla flu obo pic sax tru vio
Predicted label
25 20 15 10 5 0


Residual Data
True label
bas 29 3 0 0 0 0 0 0 0
cel 14 10 3 1 0 0 0 2 2
cla 8 1 6 1 6 3 3 4 0
flu 0 1 6 5 7 3 5 5 0
obo 0 0 7 1 16 0 0 7 1
pic 0 0 0 0 0 30 0 0 2
sax 3 0 0 0 7 1 11 10 0
tru 0 2 0 0 1 1 0 28 0
vio 0 4 2 2 4 6 3 1 10
bas cel cla flu obo pic sax tru vio
Predicted label
30 25 20 15 10 5 0

Results

We are not aiming to find the best set of features to improve the accuracy. We could improve it of course, by putting efforts in feature selection. But it is out of the scope of this project. Also, the dataset is not best suited for instrument classification because of lack of variety discussed in the Dataset section earlier. Also, the entire good-sounds dataset was not used, we picked sounds randomly. Putting more effort in creating a good subset of this dataset would have helped us in getting better results. But that again is not the scope of this project. We are interested in looking at the performance of residual and sinusoidal models and comparing them with original models.

We believed that music instrument characteristics are best captured by residual components. However, the results show something different yet interesting. The overall accuracy of original and sinusoidal models are very close at around 60% and accuracy of residual models is around 50%. But we need to observe the confusion matrices closely. Sinusoidal models did better than original models for flute(sinusoidal model accuracy: 14/32 original model accuracy:11/32) and violin(sinusoidal model accuracy:22/32 original model accuracy:11/32, almost double!). Residual models did better than original models for bas(residual model accuracy:29/32 original model accuracy:28/32), piccolo(residual model accuracy:30/32 original model accuracy:23/32) and trumpet(residual model accuracy:28/32 original model accuracy:27/32).

Based on these results, it is worthwhile to do a feature selection and better dataset creation to see what results we get. We could also run the residual models and sinusoidal models on original dataset and see if it improves performance of any instruments. Then we could combine the models and improve the overall accuracy since we have models which are specialized in identifying particular instruments and we can assign these models a priority value which will have a greater say in predicting an instrument that it specializes in.

References

Romani Picas O. Dabiri D., Serra X. "A real-time system for meauring sound goodness in instrumental sounds" 138th Audio Engineering Society Convention, Warsarw, 2015