PySciCal Project - machine learning component

This script's purpose is to:

- import converted npy music files
- visualize data
- build machine learning models using PCA method and other models

Import packages

In [7]:
```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import pylab
from array import array as pyarray
from numpy import append, array, int8, uint8, zeros
import numpy as np
import glob
import pandas as pd
from pandas import DataFrame
from itertools import chain
```

Import dataset

In [12]:
```
# uploading npy arrays in a group and storing them as one array
    # transformation is made to single array (one song) to become one di
mensional i.e. (1,9600) vs (600,16)
    # so that one row is one observation
    # label1 is type of music for 1; label2 is type of music for 2
def dataset_array(music1_folder, music2_folder, label1, label2):

    # get music type 1 loaded
    files = glob.glob(music1_folder)
    grouped1 = np.zeros([1,9601])
    for file in files:
        single1 = np.load(file)
        single1 = single1.reshape(1,9600)
        single1 = np.append(single1,label1)
        grouped1 = np.vstack([grouped1,single1])

    # get music type 2 loaded
    files = glob.glob(music2_folder)
    grouped2 = np.zeros([1,9601])
    for file in files:
        single2 = np.load(file)
        single2 = single2.reshape(1,9600)
        single2 = np.append(single2,label2)
        grouped2 = np.vstack([grouped2,single2])

    # join music1 and music2, but first remove dummy row
    grouped1 = grouped1[1:,:]
    grouped2 = grouped2[1:,:]
    dataset = np.vstack([grouped1,grouped2])

    print("Shape of dataset array is",dataset.shape)
    print("Length of", label1, "music array is",len(grouped1))
    print("Length of", label2, "music array is",len(grouped2))
    print("Number of observations in dataset array are",len(dataset))
    return(dataset)
```

In [13]:
```
dataset = dataset_array(music1_folder='/Users/pyu18/Documents/PySciCalPr
oject-master/spectral_loudness/wav_data/classical/npy_files/*.npy' ,
                        music2_folder='/Users/pyu18/Documents/PySciCalPr
oject-master/spectral_loudness/wav_data/pop/npy_files/*.npy' ,
                        label1 = 'pop', label2= 'classical')
```

```
Shape of dataset array is (33, 9601)
Length of pop music array is 16
Length of classical music array is 17
Number of observations in dataset array are 33
```

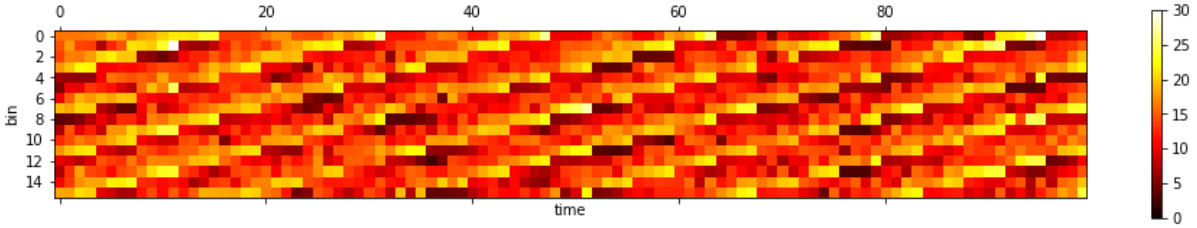
Data Exploration

Visualization of classical music

In [14]:
```
# get a sample of music1
classical = [x for x in dataset if 'classical' in x[9600]]
classical_v = np.asarray(classical[1])
classical_v = np.asarray(classical_v[:9600], dtype = float)
```

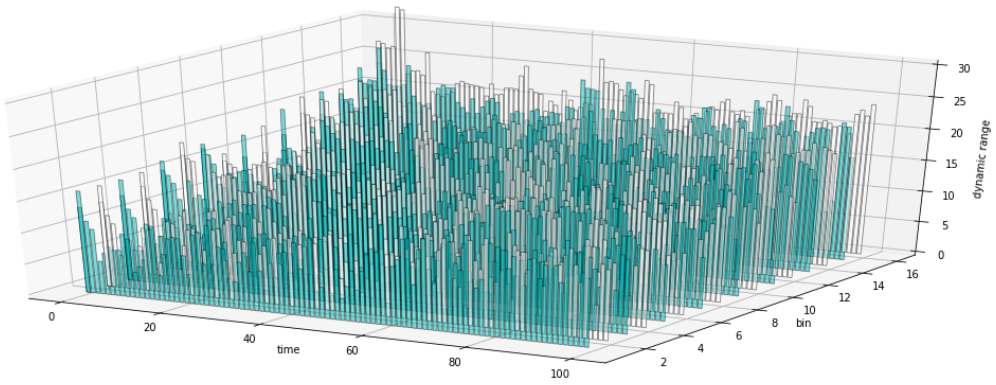
In [16]:
```
## run heatmap

print("Check Mins and Max
Values",np.min(classical_v),np.max(classical_v))
print("Check Mean Values",np.mean(classical_v))
print("")
print("Getting a 10 second slice of music:")
classical_sliced = classical_v[:1600]
classical_sliced = classical_sliced.reshape(16,100)
plt.matshow(classical_sliced, interpolation = "nearest",vmin=0, vmax=30,
 cmap="hot")
#fig.colorbar(plt.matshow(classical_sliced, interpolation = "nearest", c
map="hot"))
plt.colorbar()
plt.xlabel('time')
plt.ylabel('bin')
plt.show()
#cax = ax.matshow(data, interpolation='nearest')
#fig.colorbar(cax)
```

```
Check Mins and Max Values 2.29046945106 34.5662008785
Check Mean Values 14.762662421

Getting a 10 second slice of music:
```



In [17]:
```
# plot this in a 3d bar plot
print("Dynamic ranges of a classical piece in 10 seconds:")
fig = plt.figure(1,figsize=(20,7))
ax = fig.add_subplot(111, projection='3d')

x = np.linspace(1,1,16)
for n in range(1,100):
    x_n = np.linspace(n,n,16)
    x = list(chain(x,x_n))

z = np.linspace(1,16,16)
for n in range(1,100):
    z_1 = np.linspace(1,16,16)
    z = list(chain(z,z_1))

y = classical_sliced.reshape(1600,1)

ax.set_xlabel('time')
ax.set_ylabel('bin')
ax.set_zlabel('dynamic range')
ax.bar(x,y,z,zdir='y',color='c1', edgecolor = 'black',linewidth=.8, alph
a=.5)
ax.set_zlim3d(0,30)
#ax.plot_wireframe(x, z, y, rstride=10, cstride=10,edgecolor = 'black',l
inewidth = .5)
plt.show()
```

```
Dynamic ranges of a classical piece in 10 seconds:
```



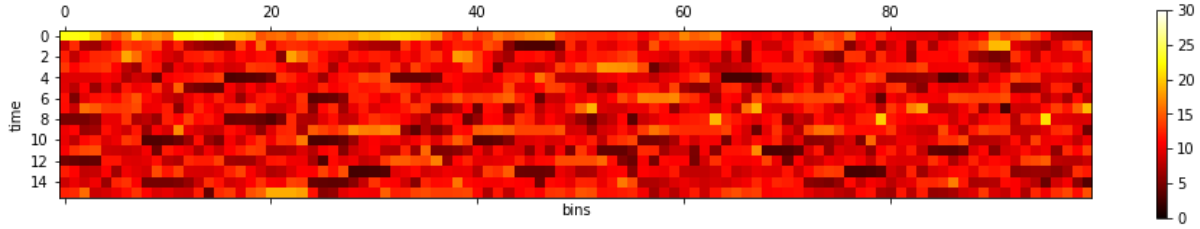
Visualization of pop music

In [18]:
```
# get a sample of music2
pop = [x for x in dataset if 'pop' in x[9600]]
pop_v = np.asarray(pop[1])
print(pop_v)
pop_v = np.asarray(pop_v[:9600], dtype = float)
```

```
['22.844698720513627' '22.815619180240063' '22.549184841531783' ...,
 '17.197433635456143' '16.424276439005602' 'pop']
```

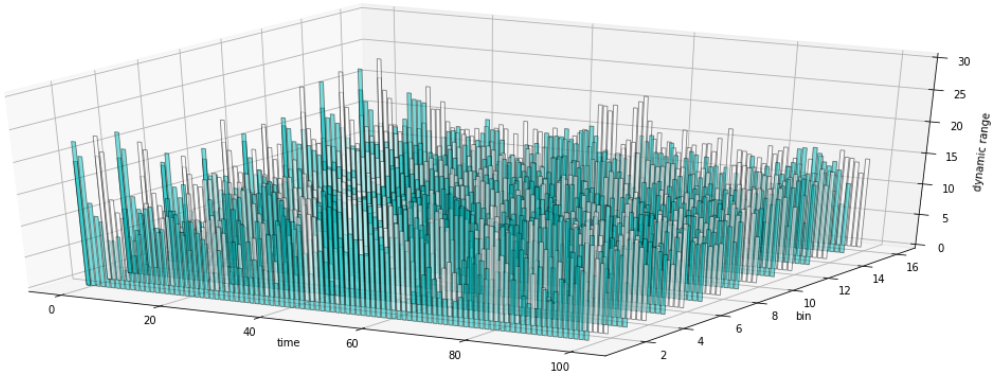
In [19]:
```
## run heatmap

print("Check shape", pop_v.shape)
print("Check Mins and Max Values",np.min(pop_v),np.max(pop_v))
print("Check Mean Values",np.mean(pop_v))
print("")
print("Getting a 10 second slice of music:")
pop_sliced = pop_v[:1600]
pop_sliced = pop_sliced.reshape(16,100)
plt.matshow(pop_sliced, interpolation = "nearest", vmin=0, vmax=30,
cmap="hot")
#plt.matshow(pop_sliced, vmin=10, vmax=90)
plt.xlabel('bins')
plt.ylabel('time')
plt.colorbar()
plt.show()
```

```
Check shape (9600,)
Check Mins and Max Values 2.1827243315 36.1477940501
Check Mean Values 11.1483522055

Getting a 10 second slice of music:
```



In [20]:
```
# plot this in a 3d bar plot
print("Dynamic ranges of a pop piece in 10 seconds:")
fig = plt.figure(1,figsize=(20,7))
ax = fig.add_subplot(111, projection='3d')

x = np.linspace(1,1,16)
for n in range(1,100):
    x_n = np.linspace(n,n,16)
    x = list(chain(x,x_n))

z = np.linspace(1,16,16)
for n in range(1,100):
    z_1 = np.linspace(1,16,16)
    z = list(chain(z,z_1))

y = pop_sliced.reshape(1600,1)

ax.set_xlabel('time')
ax.set_ylabel('bin')
ax.set_zlabel('dynamic range')
ax.bar(x,y,z,zdir='y',color='c1', edgecolor = 'black',linewidth=.8, alph
a=.5)
#ax.plot_wireframe(x, z, y, rstride=10, cstride=10,edgecolor = 'black',l
inewidth = .5)
ax.set_zlim3d(0,30)
plt.show()
```

```
Dynamic ranges of a pop piece in 10 seconds:
```



Prepare data for modelling

This involves:

a. Randomizing order of data while keeping maintaining labels

Shuffle the data before splitting data into test and training to ensure ther
e are no dependencies to order of the row

b. Splitting data between training and testing

Intend to keep data between 80:20 whereby training data is 80% of dataset an
d testing data is 20% of dataset

In [21]:
```
# Randomize data
shuffle = np.random.permutation(np.arange(dataset.shape[0]))
dataset = dataset[shuffle]
```

In [22]:
```
# Splitting data
dataset_data, dataset_labels = dataset[:, :9600], dataset[:, -1]
# Split dataset 80:20 into test and train
train_data, train_labels = dataset_data[:26], dataset_labels[:26]
test_data, test_labels = dataset_data[26:], dataset_labels[26:]
print("train_data shape is", train_data.shape, "and train_label shape i
s", train_labels.shape)
print("test_data shape is", test_data.shape, "and test_label shape is",
test_labels.shape)

## return train and test data into float
train_data = np.asarray(train_data, dtype=float)
test_data = np.asarray(test_data, dtype=float)
```

```
train_data shape is (26, 9600) and train_label shape is (26,)
test_data shape is (7, 9600) and test_label shape is (7,)
```

Perform PCA to reduce dimensions

In [23]:
```
# Import PCA
from sklearn.decomposition import PCA
```

In [24]:
```
# Create PCA instance: model
pca = PCA()

# Apply fit_transform method to training: pca-features
pca_features = pca.fit_transform(train_data)
pca_explnd_variance = pca.explained_variance_ratio_
print(pca_features.shape)
```

```
(26, 26)
```

Decide how many PCA components to select by graphing variance explained

In [25]:
```
# Each additional dimension explains less additioal variance
print("PCA explained variance:")
print(pca_explnd_variance)
```

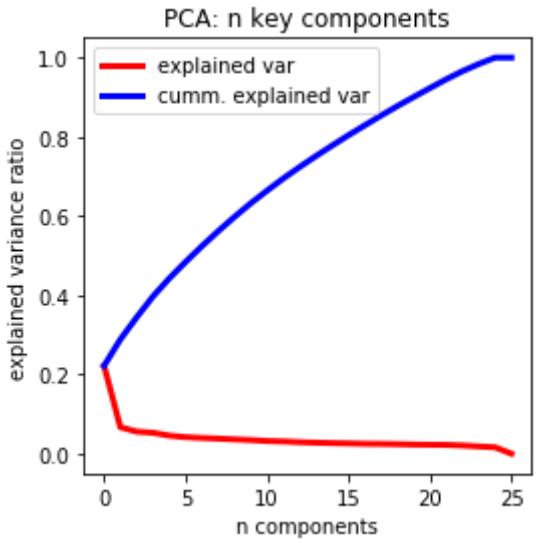
```
PCA explained variance:
[  2.21839229e-01   6.67469849e-02   5.55899306e-02   5.27565109e-02
   4.55450461e-02   4.15567393e-02   3.96376347e-02   3.79262860e-02
   3.60547205e-02   3.43742337e-02   3.18984548e-02   3.08760837e-02
   2.88323899e-02   2.76885428e-02   2.66623918e-02   2.59475926e-02
   2.49803208e-02   2.46088416e-02   2.39577305e-02   2.29741286e-02
   2.25431655e-02   2.21783812e-02   2.05988021e-02   1.81254725e-02
   1.61003859e-02   3.06761114e-31]
```

In [27]:
```
print("cum PCA explained variance:")
np.cumsum(pca_explnd_variance) # looks like 25 components are sufficient
 to explain all the variance
```

```
cum PCA explained variance:
```

Out[27]:
```
array([ 0.22183923,  0.28858621,  0.34417614,  0.39693266,  0.4424777 ,
        0.48403444,  0.52367208,  0.56159836,  0.59765308,  0.63202732,
        0.66392577,  0.69480185,  0.72363424,  0.75132279,  0.77798518,
        0.80393277,  0.82891309,  0.85352193,  0.87747966,  0.90045379,
        0.92299696,  0.94517534,  0.96577414,  0.98389961,  1.        ,
  1.        ])
```

In [28]:
```
# Plot explaineing explained and cumulative explained variance
plt.figure(1,figsize=(4,4))
plt.clf()
plt.plot(pca_explnd_variance, linewidth=3,c='r',label='explained var')
plt.plot(np.cumsum(pca_explnd_variance),  linewidth=3, c='b', label='cum
m. explained var')
plt.axis('tight')
plt.xlabel('n components')
plt.ylabel('explained variance ratio')
plt.legend()
plt.title('PCA: n key components')
plt.show()
```



Apply PCA on testing and training dataset

In [17]:
```
# apply PCA on testing and training based on recommended number of compo
nents
pca1 = PCA(n_components=25).fit(train_data)
reduced_training = pca1.transform(train_data)
reduced_testing = pca1.transform(test_data) # find a test data
print("reduced training data shape is now", reduced_training.shape)
print("reduced testing data shape is now", reduced_testing.shape)
```

```
reduced training data shape is now (26, 25)
reduced testing data shape is now (7, 25)
```

Let's start to apply models to the reduced dimension dataset!

In [29]:
```
# Import relevant packages
#from sklearn.metrics import confusion_matrix
from sklearn import svm, tree

from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import BernoulliNB, MultinomialNB, GaussianNB
from sklearn.grid_search import GridSearchCV, RandomizedSearchCV
from matplotlib.colors import LogNorm
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier,
 ExtraTreesClassifier
from sklearn.cluster import KMeans
from sklearn.mixture import GMM
```

```
/Users/pyu18/anaconda/lib/python3.6/site-packages/sklearn/cross_validat
ion.py:44: DeprecationWarning: This module was deprecated in version 0.
18 in favor of the model_selection module into which all the refactored
classes and functions are moved. Also note that the interface of the ne
w CV iterators are different from that of this module. This module will
be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
/Users/pyu18/anaconda/lib/python3.6/site-packages/sklearn/grid_search.p
y:43: DeprecationWarning: This module was deprecated in version 0.18 in
favor of the model_selection module into which all the refactored class
es and functions are moved. This module will be removed in 0.20.
  DeprecationWarning)
```

In [30]:
```
#def check_model(model, print_confusion_matrix):
def run_model(model_name):
    model = model_name()
    model.fit(reduced_training, train_labels)
    score = model.score(reduced_testing, test_labels)
    print("model:", model_name)
    print("score:", score)
    result = model.predict(reduced_testing)
    print(pd.DataFrame(data = np.transpose([test_labels,result]), column
s = ['actual','predicted']))
```

In [31]:
```
## all the models we plan to execute
def run_all_models():
    run_model(LogisticRegression)
    print("")
    run_model(svm.SVC)
    print("")
    run_model(BernoulliNB)
    print("")
    run_model(GaussianNB)
    print("")
    run_model(tree.DecisionTreeClassifier)
    print("")
    run_model(RandomForestClassifier)
    print("")
    run_model(ExtraTreesClassifier)
    print("")
    run_model(AdaBoostClassifier)
```

In [27]:
```
## running all models
run_all_models()
```

```
model: <class 'sklearn.linear_model.logistic.LogisticRegression'>
score: 1.0
      actual  predicted
0  classical  classical
1  classical  classical
2        pop        pop
3  classical  classical
4        pop        pop
5        pop        pop
6        pop        pop

model: <class 'sklearn.svm.classes.SVC'>
score: 0.428571428571
      actual  predicted
0  classical  classical
1  classical  classical
2        pop  classical
3  classical  classical
4        pop  classical
5        pop  classical
6        pop  classical

model: <class 'sklearn.naive_bayes.BernoulliNB'>
score: 0.857142857143
      actual  predicted
0  classical  classical
1  classical  classical
2        pop        pop
3  classical        pop
4        pop        pop
5        pop        pop
6        pop        pop

model: <class 'sklearn.naive_bayes.GaussianNB'>
score: 1.0
      actual  predicted
0  classical  classical
1  classical  classical
2        pop        pop
3  classical  classical
4        pop        pop
5        pop        pop
6        pop        pop

model: <class 'sklearn.tree.tree.DecisionTreeClassifier'>
score: 1.0
      actual  predicted
0  classical  classical
1  classical  classical
2        pop        pop
3  classical  classical
4        pop        pop
5        pop        pop
6        pop        pop

model: <class 'sklearn.ensemble.forest.RandomForestClassifier'>
score: 0.857142857143
```

```
      actual  predicted
0  classical  classical
1  classical  classical
2        pop  classical
3  classical  classical
4        pop        pop
5        pop        pop
6        pop        pop

model: <class 'sklearn.ensemble.forest.ExtraTreesClassifier'>
score: 1.0
      actual  predicted
0  classical  classical
1  classical  classical
2        pop        pop
3  classical  classical
4        pop        pop
5        pop        pop
6        pop        pop

model: <class 'sklearn.ensemble.weight_boosting.AdaBoostClassifier'>
score: 1.0
      actual  predicted
0  classical  classical
1  classical  classical
2        pop        pop
3  classical  classical
4        pop        pop
5        pop        pop
6        pop        pop
```