FMA: A Dataset For Music Analysis

Michaël Defferrard, Kirell Benzi, Pierre Vandergheynst, Xavier Bresson, EPFL LTS2.

Baselines

- This notebook evaluates standard classifiers from scikit-learn on the provided features.
- Moreover, it evaluates Deep Learning models on both audio and spectrograms.

In [1]:
```
import time
import os

import IPython.display as ipd
from tqdm import tqdm_notebook
import numpy as np
import pandas as pd
import keras
from sklearn.utils import shuffle
from sklearn.preprocessing import MultiLabelBinarizer, LabelEncoder, LabelBinarizer
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB,MultinomialNB
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.multiclass import OneVsRestClassifier
import warnings
warnings.filterwarnings('ignore')
import utils
```

```
WARNING:tensorflow:From C:\Users\shivd\anaconda3\Lib\site-packages\keras\src\losse
s.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated. Please u
se tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

In [2]:
```
AUDIO_DIR = os.environ.get('AUDIO_DIR')
AUDIO_DIR='./data/fma_small/'
tracks = utils.load('data/fma_metadata/tracks.csv')
features = utils.load('data/fma_metadata/features.csv')
echonest = utils.load('data/fma_metadata/echonest.csv')

np.testing.assert_array_equal(features.index, tracks.index)
assert echonest.index.isin(tracks.index).all()

tracks.shape, features.shape, echonest.shape
```

Out[2]:
```
((106574, 52), (106574, 518), (13129, 249))
```

Subset

In [3]:
```
subset = tracks.index[tracks['set', 'subset'] <= 'medium']

assert subset.isin(tracks.index).all()
assert subset.isin(features.index).all()

features_all = features.join(echonest, how='inner').sort_index(axis=1)
print('Not enough Echonest features: {}'.format(features_all.shape))
```

```
tracks = tracks.loc[subset]
features_all = features.loc[subset]

tracks.shape, features_all.shape
```

```
Not enough Echonest features: (13129, 767)
```

Out[3]:

```
((25000, 52), (25000, 518))
```

In [4]:

```
train = tracks.index[tracks['set', 'split'] == 'training']
val = tracks.index[tracks['set', 'split'] == 'validation']
test = tracks.index[tracks['set', 'split'] == 'test']

print('{} training examples, {} validation examples, {} testing examples'.format(*m

genres = list(LabelEncoder().fit(tracks['track', 'genre_top']).classes_)
#genres = list(tracks['track', 'genre_top'].unique())
print('Top genres ({}): {}'.format(len(genres), genres))
genres = list(MultiLabelBinarizer().fit(tracks['track', 'genres_all']).classes_)
print('All genres ({}): {}'.format(len(genres), genres))
```

```
19922 training examples, 2505 validation examples, 2573 testing examples
Top genres (16): ['Blues', 'Classical', 'Country', 'Easy Listening', 'Electronic',
'Experimental', 'Folk', 'Hip-Hop', 'Instrumental', 'International', 'Jazz', 'Old-T
ime / Historic', 'Pop', 'Rock', 'Soul-RnB', 'Spoken']
All genres (151): [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
19, 20, 21, 22, 25, 26, 27, 30, 31, 32, 33, 36, 37, 38, 41, 42, 43, 45, 46, 47, 4
9, 53, 58, 63, 64, 65, 66, 70, 71, 74, 76, 77, 79, 81, 83, 85, 86, 88, 89, 90, 92,
94, 97, 98, 100, 101, 102, 103, 107, 109, 111, 113, 117, 118, 125, 130, 137, 138,
166, 167, 169, 171, 172, 174, 177, 179, 180, 181, 182, 183, 184, 185, 186, 187, 18
8, 189, 214, 224, 232, 236, 240, 247, 250, 267, 286, 296, 297, 311, 314, 322, 337,
359, 360, 361, 362, 374, 378, 400, 401, 404, 428, 439, 440, 441, 442, 443, 456, 46
8, 491, 495, 502, 504, 514, 524, 538, 539, 542, 580, 602, 619, 651, 659, 695, 741,
763, 808, 810, 811, 906, 1032, 1060, 1193, 1235]
```

1 Multiple classifiers and feature sets

Todo:

- Cross-validation for hyper-parameters.
- Dimensionality reduction?

In [5]:

```
tracks['track', 'genres_all']
```

Out[5]:

```
track_id
2                      [21]
3                      [21]
5                      [21]
10                     [10]
134                    [21]
                ...
155297    [107, 18, 1235]
155298          [17, 103]
155306          [17, 103]
155307            [1, 38]
155314           [25, 12]
Name: (track, genres_all), Length: 25000, dtype: object
```

In [6]:

```
tracks['track', 'genre_top'].value_counts()
```

Out[6]:
```
(track, genre_top)
Rock                   7103
Electronic             6314
Experimental           2251
Hip-Hop                2201
Folk                   1519
Instrumental           1350
Pop                    1186
International          1018
Classical               619
Old-Time / Historic     510
Jazz                    384
Country                 178
Soul-RnB                154
Spoken                  118
Blues                    74
Easy Listening           21
Name: count, dtype: int64
```

In [7]:
```python
tracks['track', 'genres_all']
```

Out[7]:
```
track_id
2                    [21]
3                    [21]
5                    [21]
10                   [10]
134                  [21]
                ...
155297    [107, 18, 1235]
155298          [17, 103]
155306          [17, 103]
155307            [1, 38]
155314           [25, 12]
Name: (track, genres_all), Length: 25000, dtype: object
```

1.1 Pre-processing

In [8]:
```python
def pre_process(tracks, features, columns, multi_label=False, verbose=False):
    if not multi_label:
        # Assign an integer value to each genre.
        enc = LabelEncoder()
        labels = tracks['track', 'genre_top']
        #y = enc.fit_transform(tracks['track', 'genre_top'])
    else:
        # Create an indicator matrix.
        enc = MultiLabelBinarizer()
        labels = tracks['track', 'genres_all']
        #labels = tracks['track', 'genres']

    # Split in training, validation and testing sets.
    y_train = enc.fit_transform(labels[train])
    y_val = enc.transform(labels[val])
    y_test = enc.transform(labels[test])
    X_train = features.loc[train, columns].values
    X_val = features.loc[val, columns].values
    X_test = features.loc[test, columns].values

    X_train, y_train = shuffle(X_train, y_train, random_state=42)

    # Standardize features by removing the mean and scaling to unit variance.
    scaler = StandardScaler(copy=False)
    scaler.fit_transform(X_train)
    scaler.transform(X_val)
```

```
    scaler.transform(X_test)

    return y_train, y_val, y_test, X_train, X_val, X_test
```

1.2 Single genre

In [9]:

```
def test_classifiers_features(classifiers, feature_sets, multi_label=False):
    columns = list(classifiers.keys()).insert(0, 'dim')
    scores = pd.DataFrame(columns=columns, index=feature_sets.keys())
    times = pd.DataFrame(columns=classifiers.keys(), index=feature_sets.keys())
    for fset_name, fset in tqdm_notebook(feature_sets.items(), desc='features'):
        y_train, y_val, y_test, X_train, X_val, X_test = pre_process(tracks, featur
        scores.loc[fset_name, 'dim'] = X_train.shape[1]
        for clf_name, clf in classifiers.items():  # tqdm_notebook(classifiers.iten
            t = time.process_time()
            clf.fit(X_train, y_train)
            score = clf.score(X_test, y_test)
            scores.loc[fset_name, clf_name] = score
            times.loc[fset_name, clf_name] = time.process_time() - t
    return scores, times

def format_scores(scores):
    def highlight(s):
        is_max = s == max(s[1:])
        return ['background-color: yellow' if v else '' for v in is_max]
    scores = scores.style.apply(highlight, axis=1)
    return scores.format('{:.2%}', subset=pd.IndexSlice[:, scores.columns[1]:])
```

In [10]:

```
classifiers = {'kNN': KNeighborsClassifier(n_neighbors=200),'NB': GaussianNB()}


feature_sets = {
#    'echonest_audio': ('echonest', 'audio_features'),
#    'echonest_social': ('echonest', 'social_features'),
#    'echonest_temporal': ('echonest', 'temporal_features'),
#    'echonest_audio/social': ('echonest', ('audio_features', 'social_features')),
#    'echonest_all': ('echonest', ('audio_features', 'social_features', 'temporal_f
}
for name in features.columns.levels[0]:
    feature_sets[name] = name
feature_sets.update({
    'mfcc/contrast': ['mfcc', 'spectral_contrast'],
    'mfcc/contrast/chroma': ['mfcc', 'spectral_contrast', 'chroma_cens'],
    'mfcc/contrast/centroid': ['mfcc', 'spectral_contrast', 'spectral_centroid'],
    'mfcc/contrast/chroma/centroid': ['mfcc', 'spectral_contrast', 'chroma_cens', '
    'mfcc/contrast/chroma/centroid/tonnetz': ['mfcc', 'spectral_contrast', 'chroma_
    'mfcc/contrast/chroma/centroid/zcr': ['mfcc', 'spectral_contrast', 'chroma_cens
    'all_non-echonest': list(features.columns.levels[0])
})

scores, times = test_classifiers_features(classifiers, feature_sets)

ipd.display(format_scores(scores))
ipd.display(times.style.format('{:.4f}'))
```

```
features:   0%|          | 0/18 [00:00<?, ?it/s]
```

	dim	kNN	NB
chroma_cens	84.000000	37.50%	9.99%
chroma_cqt	84.000000	40.03%	1.55%
chroma_stft	84.000000	43.92%	4.20%
mfcc	140.000000	54.99%	41.86%
rmse	7.000000	38.52%	11.78%
spectral_bandwidth	7.000000	45.39%	36.18%
spectral_centroid	7.000000	45.36%	33.31%
spectral_contrast	49.000000	49.55%	39.41%
spectral_rolloff	7.000000	46.25%	28.49%
tonnetz	42.000000	37.31%	22.31%
zcr	7.000000	44.73%	30.39%
mfcc/contrast	189.000000	55.31%	44.03%
mfcc/contrast/chroma	273.000000	53.13%	39.02%
mfcc/contrast/centroid	196.000000	55.23%	43.76%
mfcc/contrast/chroma/centroid	280.000000	53.01%	38.87%
mfcc/contrast/chroma/centroid/tonnetz	322.000000	52.62%	39.06%
mfcc/contrast/chroma/centroid/zcr	287.000000	53.01%	38.90%
all_non-echonest	518.000000	51.77%	9.91%

	kNN	NB
chroma_cens	7.1875	0.0625
chroma_cqt	5.9531	0.0781
chroma_stft	6.3438	0.0312
mfcc	6.2656	0.1094
rmse	0.7500	0.0000
spectral_bandwidth	0.6094	0.0156
spectral_centroid	0.4531	0.0000
spectral_contrast	5.7969	0.0156
spectral_rolloff	0.3750	0.0000
tonnetz	5.6875	0.0000
zcr	0.4844	0.0156
mfcc/contrast	7.3750	0.0938
mfcc/contrast/chroma	8.6406	0.1094
mfcc/contrast/centroid	7.2500	0.0625
mfcc/contrast/chroma/centroid	8.1719	0.1875
mfcc/contrast/chroma/centroid/tonnetz	9.3906	0.1719
mfcc/contrast/chroma/centroid/zcr	8.8906	0.1562
all_non-echonest	13.5625	0.0938

1.3 Multiple genres

Todo:

- Ignore rare genres? Count them higher up in the genre tree? On the other hand it's not much tracks.

In [13]:
```
classifiers = {
    #LogisticRegression(),
    'LR': OneVsRestClassifier(LogisticRegression(n_jobs=-1)),
    'MLP': MLPClassifier(max_iter=700),
}

feature_sets = {
#    'echonest_audio': ('echonest', 'audio_features'),
#    'echonest_temporal': ('echonest', 'temporal_features'),
    'mfcc': 'mfcc',
    'mfcc/contrast/chroma/centroid/tonnetz': ['mfcc', 'spectral_contrast', 'chroma_
    'mfcc/contrast/chroma/centroid/zcr': ['mfcc', 'spectral_contrast', 'chroma_cens
}

scores, times = test_classifiers_features(classifiers, feature_sets, multi_label=Tr

ipd.display(format_scores(scores))
ipd.display(times.style.format('{:.4f}'))
```

```
features:   0%|          | 0/3 [00:00<?, ?it/s]
```

	dim	LR	MLP
mfcc	140.000000	11.23%	12.09%
mfcc/contrast/chroma/centroid/tonnetz	322.000000	12.90%	8.36%
mfcc/contrast/chroma/centroid/zcr	287.000000	13.06%	9.76%

	LR	MLP
mfcc	1.6562	1742.2500
mfcc/contrast/chroma/centroid/tonnetz	6.2812	1437.9844
mfcc/contrast/chroma/centroid/zcr	6.1719	969.3906

In [ ]:

```

```