**Tutorial for the Million Song Dataset**

Copyright 2011 T. Bertin-Mahieux, All Rights Reserved

This tutorial will walk you through a quick experiment using the Million Song Dataset (MSD). We will actually be working on the 10K songs subset for speed issues, but the code should transpose seamlessly.

In this tutorial, we do simple metadata analysis. We look at which artist has the most songs by iterating over the whole dataset and using an SQLite database.

You need to have the MSD code downloaded from GITHUB. See the MSD website for details: [http://labrosa.ee.columbia.edu/millionsong/](http://labrosa.ee.columbia.edu/millionsong/)

If you have any questions regarding the dataset or this tutorial, please first take a look at the website. Send us an email if you haven’t found the answer.

Note: this tutorial is developed using Python 2.6 on an Ubuntu machine. PDF created using 'pyreport'.

```python
### usual imports
import os
import sys
import time
import glob
import datetime
import sqlite3
import numpy as np  # get it at: http://numpy.scipy.org/

### path to the Million Song Dataset subset (uncompressed)
# CHANGE IT TO YOUR LOCAL CONFIGURATION
msd_subset_path = '/home/thierry/Desktop/MillionSongSubset'
msd_subset_data_path = os.path.join(msd_subset_path, 'data')
msd_subset_addf_path = os.path.join(msd_subset_path, 'AdditionalFiles')
assert os.path.isdir(msd_subset_path), 'wrong path'  # sanity check

### path to the Million Song Dataset code
# CHANGE IT TO YOUR LOCAL CONFIGURATION
msd_code_path = '/home/thierry/Columbia/MSongsDB'
assert os.path.isdir(msd_code_path), 'wrong path'  # sanity check

# we add some paths to python so we can import MSD code  
# Ubuntu: you can change the environment variable PYTHONPATH  
# in your .bashrc file so you do not have to type these lines
sys.path.append(os.path.join(msd_code_path, 'PythonSrc'))

### imports specific to the MSD
import hdf5_getters as GETTERS

### the following function simply gives us a nice string for  
# a time lag in seconds
```
```python
def strftimeDelta(starttime, stoptime):
    return str(datetime.timedelta(seconds=stoptime - starttime))

# we define this very useful function to iterate the files
def apply_to_all_files(basedir, func=lambda x: x, ext='.h5'):
    """
    From a base directory, go through all subdirectories,
    find all files with the given extension, apply the
    given function 'func' to all of them.
    If no 'func' is passed, we do nothing except counting.
    INPUT
    basedir - base directory of the dataset
    func - function to apply to all filenames
    ext - extension, .h5 by default
    RETURN
    number of files
    """
    cnt = 0
    # iterate over all files in all subdirectories
    for root, dirs, files in os.walk(basedir):
        files = glob.glob(os.path.join(root, '*'+ext))
        # count files
        cnt += len(files)
        # apply function to all files
        for f in files:
            func(f)
    return cnt

# we can now easily count the number of files in the dataset
print 'number of song files:', apply_to_all_files(msd Subset_data_path)

number of song files: 10000

# let's now get all artist names in a set(). One nice property:
# if we enter many times the same artist, only one will be kept.
all_artist_names = set()

# we define the function to apply to all files
def func_to_get_artist_name(filename):
    """
    This function does 3 simple things:
    - open the song file
    - get artist ID and put it
    - close the file
    """
    h5 = GETTERS.open_h5_file_read(filename)
    artist_name = GETTERS.get_artist_name(h5)
    all_artist_names.add( artist_name )
    h5.close()

# let's apply the previous function to all files
# we'll also measure how long it takes
print 'all artist names extracted in:', strftimeDelta(t1, t2)
```

```python
# let's see some of the content of 'all_artist_names'
print 'found', len(all_artist_names), 'unique artist names'

found 4412 unique artist names

for k in range(5):
    print list(all_artist_names)[k]

    Groundhogs
    Pale Forest
    The Real Kids
    JennyAnyKind
    Aswad

# this is too long, and the work of listing artist names has already
# been done. Let's redo the same task using an SQLite database.
# We connect to the provided database: track metadata.db
conn = sqlite3.connect(os.path.join(msd_subset_addf_path, 'subset_track_metadata.db'))

# we build the SQL query
q = "SELECT DISTINCT artist_name FROM songs"
# we query the database

all_artist_names_sqlite = res.fetchall()

t1 = time.time()
res = conn.execute(q)
all_artist_names_sqlite = res.fetchall()
t2 = time.time()
print 'all artist names extracted (SQLite) in:', strftime(t1-t2)

al l artist names extracted (SQLite) in: 0:00:00.011670

# we close the connection to the database
conn.close()

# let's see some of the content
for k in range(5):
    print all_artist_names_sqlite[k][0]

!!!
(hed) p.e.
089 Clique feat. Minnesota Snipe & Skinny Cueball
089 Clique feat. Prophet
1. Futurologischer Congress

# now, let's find the artist that has the most songs in the dataset
# what we want to work with is artist ID, not artist names. Some artists
# have many names, usually because the song is "featuring someone else"
conn = sqlite3.connect(os.path.join(msd_subset_addf_path, 'subset_track_metadata.db'))

q = "SELECT DISTINCT artist_id FROM songs"
res = conn.execute(q)
all_artist_ids = map(lambda x: x[0], res.fetchall())
conn.close()

# The Echo Nest artist id look like:
for k in range(4):
    print all_artist_ids[k]

AR0092111187B989185
AR00A6H1187FB5402A
AR00LNI1187FB444A5
AR00MBZ1187B9B5DB1
```
# let's count the songs from each of these artists.
# We will do it first by iterating over the dataset.
# we prepare a dictionary to count files
files_per_artist = {}
for aid in all_artist_ids:
    files_per_artist[aid] = 0

# we prepare the function to check artist id in each file
def func_to_count_artist_id(filename):
    """
    This function does 3 simple things:
    - open the song file
    - get artist ID and put it
    - close the file
    """
    h5 = GETTERS.open_h5_file_read(filename)
    artist_id = GETTERS.get_artist_id(h5)
    files_per_artist[artist_id] += 1
    h5.close()

# we apply this function to all files
apply_to_all_files(msd_subset_data_path, func=func_to_count_artist_id)

# the most popular artist (with the most songs) is:
most_pop_id = sorted(files_per_artist, key=files_per_artist.__getitem__, reverse=True)[0]
print most_pop_id, 'has', files_per_artist[most_pop_id], 'songs.'

AROIH0I122988FEB8E has 13 songs.

# of course, it is more fun to have the name(s) of this artist
# let's get it using SQLite
conn = sqlite3.connect(os.path.join(msd_subset_addf_path, 'subset_track_metadata.db'))
q = "SELECT DISTINCT artist_name FROM songs WHERE artist_id='AROIH0I122988FEB8E'"
res = conn.execute(q)
pop_artist_names = map(lambda x: x[0], res.fetchall())
conn.close()
print 'name(s) of the most popular artist:', pop_artist_names

name(s) of the most popular artist: ['Mario Rosenstock']

# let's redo all this work in SQLite in a few seconds
t1 = time.time()
conn = sqlite3.connect(os.path.join(msd_subset_addf_path, 'subset_track_metadata.db'))
q = "SELECT DISTINCT artist_id, artist_name, Count(track_id) FROM songs"
q += " GROUP BY artist_id"
res = conn.execute(q)
pop_artists = res.fetchall()
conn.close()
t2 = time.time()
print 'found most popular artist in', strtimedelta(t1, t2)

found most popular artist in 0:00:00.055763

print sorted(pop_artists, key=lambda x: x[2], reverse=True)[0]

(u'AROIHOI12988FEBSE', u'Mario Rosenstock', 13)