APPENDIX

In this Appendix we show our source code explicitly.

A. *get_urls.py*

```
import urllib2
import re
import sys


search_pattern = 'href="http://www.movie-list.com/trailers/(\S+)">/trailers/\S+</a>'

# Search through the file and find all movie title strings
filename = 'archive.php'
title_strings = []
file = open(filename, 'r')
lines = file.readlines()

for line in lines:
    titles = re.findall(search_pattern, line)
    if titles:
        # Found titles; add them!
        print "Adding %s titles" % len(titles)
        for title in titles:
            title_strings.append(title)

# Now, we have some titles, so let's write them all out to a file as urls
filename = 'urls.txt'
url_prefix = 'http://www.movie-list.com/trailers/'

with open(filename, 'w') as file:
    for title_string in title_strings:
        file.write(url_prefix + title_string + '\n')
```

B. *get_metadata.py*

```
import urllib2
import re
import sys
import time
import csv

movie_title_search_pattern = '<h1 itemprop="name">(.+)</h1>'
flv_file_search_pattern = 'src="(http://videos.movie-list.com/flvplayer.swf\?file=
    http://cdn.movie-list.com/flvideo/\S+.flv)"'
mov_file_search_pattern1 = 'HREF="(\S+\.mov)">'
mov_file_search_pattern2 = 'href="(\S+\.mov)">'
mp4_file_search_pattern = 'file: "(http://cdn.movie-list.com/hd/\S+.mp4)"'
genre_search_pattern = 'itemprop="genre">([A-Za-z0-9 ,]+)</span>'
release_date_search_pattern = 'Release Date: <span style="color:#181818; font-size
    :12px;">(.+)</span></span>'
imdb_search_pattern = 'href=[\'"](http:\/\/[w]*\.*imdb\.com\/title\/[A-Za-z0-9]+\/)
    [\'"]'
trailer_specific_file_search_pattern = 'Trailer</span>.*file: "(.*hd/.*?mp4)"'

# Grab URLs from file
urls_filename = 'urls.txt'
```

```
with open(urls_filename, 'r') as url_file:
    lines = url_file.readlines()

    # Set up a list of movies in which to put all of the movie data
    movie_list = []

    for i, line in enumerate(lines):
        # Set up a dictionary for each movie in which to store its data
        movie_dict = {}

        # Strip out surrounding whitespace to grab the url
        url = line.strip()

        print "Processing URL %s of %s (%s)" % (i, len(lines), url)

        # Grab the webpage data
        webpage_data = urllib2.urlopen(url).read()

        movie_dict['webpage_url'] = url

        # Use regexes to parse out lots of fun data
        movie_title_matches = re.findall(movie_title_search_pattern, webpage_data)
        if len(movie_title_matches) > 0:
            movie_dict['movie_title'] = movie_title_matches[0]

        flv_file_matches = re.findall(flv_file_search_pattern, webpage_data)
        if len(flv_file_matches) > 0:
            movie_dict['flv_files'] = flv_file_matches

        mov_file_matches1 = re.findall(mov_file_search_pattern1, webpage_data)
        if len(mov_file_matches1) > 0:
            movie_dict['mov1_files'] = mov_file_matches1

        mov_file_matches2 = re.findall(mov_file_search_pattern2, webpage_data)
        if len(mov_file_matches2) > 0:
            movie_dict['mov2_files'] = mov_file_matches2

        mp4_file_matches = re.findall(mp4_file_search_pattern, webpage_data)
        if len(mp4_file_matches) > 0:
            movie_dict['mp4_files'] = mp4_file_matches

        genre_matches = re.findall(genre_search_pattern, webpage_data)
        if len(genre_matches) > 0:
            movie_dict['genre'] = genre_matches[0]

        release_date_matches = re.findall(release_date_search_pattern, webpage_data)
        if len(release_date_matches) > 0:
            movie_dict['release_date'] = release_date_matches[0]

        imdb_url_matches = re.findall(imdb_search_pattern, webpage_data)
        if len(imdb_url_matches) > 0:
            movie_dict['imdb_url'] = imdb_url_matches[0]

        trailer_specific_file_match = re.findall(
            trailer_specific_file_search_pattern, webpage_data, re.DOTALL)
        if len(trailer_specific_file_match) > 0:
            movie_dict['trailer_specific_file'] = trailer_specific_file_match[0]
```

```
        if (len(mp4_file_matches) + len(mov_file_matches2) + len(mov_file_matches1)
            + len(flv_file_matches) == 0):
            # No videos found, so we'll skip adding this one to the dictionary
            pass
        else:
            # Store this dictionary in the list of movies
            movie_list.append(movie_dict)

        # A sleep command just so the website doesn't think I'm DOSing it...
        time.sleep(0.3)

# Write movie list into a csv
possible_keys = ['movie_title', 'webpage_url', 'genre', 'release_date', 'imdb_url',
    'flv_files', 'mov1_files', 'mov2_files', 'mp4_files', 'trailer_specific_file']
csv_filename = 'movies.csv'
with open(csv_filename, 'wb') as csv_file:
    writer = csv.writer(csv_file)
    writer.writerow(possible_keys)
    for movie in movie_list:
        # Make sure everything's in the right order...
        value_list = []
        for key in possible_keys:
            if key in movie:
                value_list.append(movie[key])
            else:
                # Blank cell
                value_list.append('')
        writer.writerow(value_list)
```

C. download_videos.py

```
import urllib
import csv
import sys
import os
import time
import re
import unicodedata

def download_video(url, movie_title, i):
    print "%s: Downloading trailer at %s (%s)" % (i, url, movie_title)

    video_directory = 'video_files'

    # Get filename
    filename = url.split('/')[-1]

    # Before we make a directory with this name, let's massage it into something
        that
    # can definitely be a valid directory name
    movie_title = unicodedata.normalize('NFKD', unicode(movie_title)).encode('ascii'
        , 'ignore')
    movie_title = unicode(re.sub('[^\w\s-]', '', movie_title).strip())
    movie_title = unicode(re.sub('[-\s]+', '-', movie_title).strip())
    movie_title = str(movie_title)

    if not os.path.exists(video_directory + '/' + movie_title):
        os.makedirs(video_directory + '/' + movie_title)
```

```
    # Open the url and download into the movie directory
    try:
        urllib.urlretrieve(url, video_directory + '/' + movie_title + '/' + filename
            )

    # Handle errors
    except Exception as e:
        print "Error: ", e, url

# Open .csv file
csv_filename = 'movies.csv'
with open(csv_filename, 'rb') as csvfile:
    reader = csv.DictReader(csvfile)
    for i, row in enumerate(reader):
        # Only download if there's an .mp4 file available
        if row['trailer_specific_file']:
            # Take the first available URL
            video_url = row['trailer_specific_file']

            # Grab the movie title, too
            title = row['movie_title']

            # Download!
            download_video(video_url, title, i)

    # Introduce just a little bit of delay, so we're not spamming
    # the server *quite* so much
    time.sleep(1)
```

D. add_move_data.py

```
import csv
import unicodedata
import re
import sys

# Formats a title so that it can be used as a directory name
def format_title_for_file_system(movie_title):
    try:
        movie_title = unicodedata.normalize('NFKD', unicode(movie_title)).encode('
            ascii', 'ignore')
        movie_title = unicode(re.sub('[^\w\s-]', '', movie_title).strip())
        movie_title = unicode(re.sub('[-\s]+', '-', movie_title).strip())
        movie_title = str(movie_title)
    except UnicodeDecodeError:
        print "Error parsing movie title:", movie_title
        sys.exit()

    return movie_title

# Movie metadata csv file
csv_filename = 'movies.csv'
# File containing various metrics from video analysis
video_analysis_filename = 'video_analysis.txt'
audio_analysis_filename = 'audio_analysis.txt'

# File to hold all of the new, coalesced data
final_csv_filename = 'more_trailer_data.csv'
```

```
# First, let's ingest the video analysis file into a dictionary
analysis_dict = {}

with open(video_analysis_filename, 'r') as analysis_file:
    lines = analysis_file.readlines()

while (len(lines) > 20):
    # Get the lines for a single movie
    movie_lines = lines[:21]

    # Parse data
    title = movie_lines[0].split(':')[-1].strip()
    num_frames = int(movie_lines[1].split(':')[-1])
    total_time = float(movie_lines[2].split(':')[-1])
    avg_intensity = float(movie_lines[3].split(':')[-1])
    avg_color_r = float(movie_lines[4].split(':')[-1].strip().split()[0])
    avg_color_g = float(movie_lines[4].split(':')[-1].strip().split()[1])
    avg_color_b = float(movie_lines[4].split(':')[-1].strip().split()[2])
    mean_shot_length = float(movie_lines[5].split(':')[-1])
    std_dev_shot_length = float(movie_lines[6].split(':')[-1])
    max_shot_length = float(movie_lines[7].split(':')[-1])
    min_shot_length = float(movie_lines[8].split(':')[-1])
    num_shots = int(movie_lines[9].split(':')[-1])
    stddev_color_with_letterbox_r = float(movie_lines[10].split(':')[-1].strip().
        split()[0])
    stddev_color_with_letterbox_g = float(movie_lines[10].split(':')[-1].strip().
        split()[1])
    stddev_color_with_letterbox_b = float(movie_lines[10].split(':')[-1].strip().
        split()[2])
    detail_score_mean = float(movie_lines[11].split(':')[-1])
    detail_score_std_dev = float(movie_lines[12].split(':')[-1])
    detail_score_max = float(movie_lines[13].split(':')[-1])
    detail_score_min = float(movie_lines[14].split(':')[-1])
    dark_scene_mean_length = float(movie_lines[15].split(':')[-1])
    dark_scene_length_std_dev = float(movie_lines[16].split(':')[-1])
    dark_scene_length_max = float(movie_lines[17].split(':')[-1])
    dark_scene_length_min = float(movie_lines[18].split(':')[-1])
    dark_scene_count = int(movie_lines[19].split(':')[-1])
    dark_scene_percentage = float(movie_lines[20].split(':')[-1])

    # Make a dictionary for just this movie
    movie_dict = {}
    movie_dict['title'] = title
    movie_dict['num_frames'] = num_frames
    movie_dict['total_time'] = total_time
    movie_dict['avg_intensity'] = avg_intensity
    movie_dict['avg_color_r'] = avg_color_r
    movie_dict['avg_color_g'] = avg_color_g
    movie_dict['avg_color_b'] = avg_color_b
    movie_dict['mean_shot_length'] = mean_shot_length
    movie_dict['std_dev_shot_length'] = std_dev_shot_length
    movie_dict['max_shot_length'] = max_shot_length
    movie_dict['min_shot_length'] = min_shot_length
    movie_dict['num_shots'] = num_shots
    movie_dict['stddev_color_with_letterbox_r'] = stddev_color_with_letterbox_r
    movie_dict['stddev_color_with_letterbox_g'] = stddev_color_with_letterbox_g
    movie_dict['stddev_color_with_letterbox_b'] = stddev_color_with_letterbox_b
    movie_dict['detail_score_mean'] = detail_score_mean
```

```
    movie_dict['detail_score_std_dev'] = detail_score_std_dev
    movie_dict['detail_score_max'] = detail_score_max
    movie_dict['detail_score_min'] = detail_score_min
    movie_dict['dark_scene_mean_length'] = dark_scene_mean_length
    movie_dict['dark_scene_length_std_dev'] = dark_scene_length_std_dev
    movie_dict['dark_scene_length_max'] = dark_scene_length_max
    movie_dict['dark_scene_length_min'] = dark_scene_length_min
    movie_dict['dark_scene_count'] = dark_scene_count
    movie_dict['dark_scene_percentage'] = dark_scene_percentage

    # Add this dictionary to the dictionary of all movies
    analysis_dict[title] = movie_dict

    # Remove this movie (and the trailing blank line) from our list
    lines = lines[22:]

with open(audio_analysis_filename, 'r') as analysis_file:
    lines = analysis_file.readlines()

while (len(lines) > 7):
    # Get the lines for a single movie
    movie_lines = lines[:7]

    # Parse data
    title = movie_lines[0].split(':')[-1].strip()
    mean_volume = float(movie_lines[1].split(':')[-1].strip())
    std_dev_volume = float(movie_lines[2].split(':')[-1].strip())
    min_volume = float(movie_lines[3].split(':')[-1].strip())
    max_volume = float(movie_lines[4].split(':')[-1].strip())
    sudden_rise_count_per_cut = float(movie_lines[5].split(':')[-1].strip())
    sudden_fall_count_per_cut = float(movie_lines[6].split(':')[-1].strip())

    # Make a dictionary for just this movie
    movie_dict = {}
    movie_dict['title'] = title
    movie_dict['mean_volume'] = mean_volume
    movie_dict['std_dev_volume'] = std_dev_volume
    movie_dict['min_volume'] = min_volume
    movie_dict['max_volume'] = max_volume
    movie_dict['sudden_rise_count_per_cut'] = sudden_rise_count_per_cut
    movie_dict['sudden_fall_count_per_cut'] = sudden_fall_count_per_cut

    # Add this dictionary to the dictionary of all movies
    if title in analysis_dict:
        for key in movie_dict.keys():
            # Add everything to the analysis_dict entry for this movie...
            # except for the title, which has already been added!
            if key != 'title':
                analysis_dict[title]['mean_volume'] = mean_volume
                analysis_dict[title]['std_dev_volume'] = std_dev_volume
                analysis_dict[title]['min_volume'] = min_volume
                analysis_dict[title]['max_volume'] = max_volume
                analysis_dict[title]['sudden_rise_count_per_cut'] =
                    sudden_rise_count_per_cut
                analysis_dict[title]['sudden_fall_count_per_cut'] =
                    sudden_fall_count_per_cut
    else:
        analysis_dict[title] = movie_dict
```

```
    # Remove this movie (and the trailing blank line) from our list
    lines = lines[8:]

# Now, we've accumulated a dictionary of all of the movies!
# Next, let's read in the old metadata csv file
with open(csv_filename, 'r') as csv_file:
    reader = csv.DictReader(csv_file)
    for row in reader:
        # Parse out the data for each movie
        nice_title = row['movie_title']
        webpage_url = row['webpage_url']
        genre = row['genre']
        release_date = row['release_date']
        imdb_url = row['imdb_url']

        # Now, we'll get the version of the "nicely formatted title" that will
        # match the version we pulled from the other file
        not_as_nice_title = format_title_for_file_system(nice_title)

        # Now, we'll check to see if we match anything in the analysis dictionary,
        # and if so, we'll add to it!
        if not_as_nice_title in analysis_dict:
            # A match! Let's add our data for this movie...
            analysis_dict[not_as_nice_title]['original_format_title'] = nice_title
            analysis_dict[not_as_nice_title]['movie_list_webpage_url'] = webpage_url
            analysis_dict[not_as_nice_title]['genre'] = genre
            analysis_dict[not_as_nice_title]['release_date'] = release_date
            analysis_dict[not_as_nice_title]['imdb_url'] = imdb_url

# # Print out some test data
# print analysis_dict[analysis_dict.keys()[0]]
# print analysis_dict[analysis_dict.keys()[1]]
# print analysis_dict[analysis_dict.keys()[2]]

# Now, dump everything into a new .csv file!
with open(final_csv_filename, 'wb') as csv_file:
    fieldnames = ['title', 'num_frames', 'total_time', 'avg_intensity',
        'avg_color_r', 'avg_color_g', 'avg_color_b', 'mean_shot_length',
        'std_dev_shot_length', 'max_shot_length', 'min_shot_length',
        'num_shots', 'stddev_color_with_letterbox_r', 'stddev_color_with_letterbox_g
            ',
        'stddev_color_with_letterbox_b', 'detail_score_mean', 'detail_score_std_dev'
            , 'detail_score_max',
        'detail_score_min' , 'dark_scene_mean_length', 'dark_scene_length_std_dev',
            'dark_scene_length_max',
        'dark_scene_length_min', 'dark_scene_count', 'dark_scene_percentage', '
            original_format_title', 'movie_list_webpage_url',
        'genre', 'release_date', 'imdb_url', 'mean_volume', 'std_dev_volume',
        'min_volume', 'max_volume', 'sudden_rise_count_per_cut', '
            sudden_fall_count_per_cut']
    writer = csv.DictWriter(csv_file, fieldnames=fieldnames)

    writer.writeheader()

    for key in analysis_dict.keys():
        writer.writerow(analysis_dict[key])
```

E. extract_mp3s.py

```
import os
import sys

video_dir = 'C:\\Users\\Noel_K\\git_repository\\video-analysis-amath-582\\
    video_processing\\video_files'
mp3_dir = 'C:\\Users\\Noel_K\\git_repository\\video-analysis-amath-582\\
    video_processing\\mp3s'

def save_audio(full_pathname, movie_title, filename):
    filename_stem = '.'.join(filename.split('.')[:-1])

    print "Converting %s to %s" % (full_pathname, os.path.join(mp3_dir,
        filename_stem + '.mp3'))
    os.system('C:\\ffmpeg\\bin\\ffmpeg.exe -i "%s" "%s"' % (full_pathname, os.path.
        join(mp3_dir, movie_title + '.mp3')))

def main():
    # Go through all directories
    for dirname, dirnames, filenames in os.walk(video_dir):
        # print dirname
        # print dirnames
        # print filenames
        # print '---'

        # Only look at non-empty directories
        if len(filenames) > 0:
            for filename in filenames:
                movie_title = dirname.split('\\')[-1]
                save_audio(os.path.join(dirname, filename), movie_title, filename)

if __name__ == '__main__':
    main()
```

F. clip_mp3s.py

```
import os
import sys
from pydub import AudioSegment
AudioSegment.converter = r"C:\\ffmpeg\\bin\\ffmpeg.exe"

mp3_dir = 'C:\\Users\\Noel_K\\git_repository\\video-analysis-amath-582\\
    video_processing\\mp3s'
clipped_mp3_dir = 'C:\\Users\\Noel_K\\git_repository\\video-analysis-amath-582\\
    video_processing\\clipped'

clip_size_ms = 10000

def clip_and_save_audio(full_pathname, filename):

    sound = AudioSegment.from_mp3(full_pathname)

    # len() and slicing are in milliseconds
    halfway_point = len(sound) / 2
    start_point = halfway_point - clip_size_ms / 2
    stop_point = halfway_point + clip_size_ms / 2
    clip_samples = sound[start_point:stop_point]
```

```
    filename_stem = '.'.join(filename.split('.')[:-1])
    print "Saving %s as %s" % (full_pathname, os.path.join(clipped_mp3_dir,
        filename_stem + '.mp3'))

    clip_samples.export(os.path.join(clipped_mp3_dir, filename_stem + '.mp3'),
        format="mp3")

    print full_pathname

def main():
    # Go through all directories
    for dirname, dirnames, filenames in os.walk(mp3_dir):
        # print dirname
        # print dirnames
        # print filenames
        # print '---'

        # Only look at non-empty directories
        if len(filenames) > 0:
            for filename in filenames:
                clip_and_save_audio(os.path.join(dirname, filename), filename)

if __name__ == '__main__':
    main()
```

G. analyze_videos.py

```
import cv2
import os
import sys
import numpy as np
import csv
from moviepy.editor import VideoFileClip, AudioClip
import time

video_analysis_mode = True
video_dir = 'F:\\582 videos\\video_files'

if video_analysis_mode:
    analysis_results_file = 'video_analysis.txt'
else:
    analysis_results_file = 'audio_analysis.txt'

# Get a frame from the current video source
def getFrame(cap):
    _, frame = cap.read()
    return frame

def is_video_already_analyzed(movie_title):
    # print movie_title
    # Check in the file and make sure we don't already have an entry for this
    # with open('video_analysis.txt', 'r') as infile:
    with open(analysis_results_file, 'r') as infile:
        lines = infile.readlines()
        for line in lines:
            if 'movie_title: ' + str(movie_title) in line:
                return True
            else:
                pass
```

```
        return False

# Get the sound amplitude statistics for the video
# (mean, standard deviation, max, and minimum)
def analyze_sound(filename):
    print 'Analyzing audio for', filename

    # Load video file
    clip = VideoFileClip(filename)

    # Make a lambda function for breaking a video clip into sound arrays
    cut = lambda i: clip.audio.subclip(i, i+1).to_soundarray(fps=44100, nbytes=4)

    # Make a lambda function for grabbing the volume of a sound array
    volume = lambda array: np.sqrt(((1.0*array)**2).mean())
    # Grab the volumes for this video file
    volumes = [volume(cut(i)) for i in range(0, int(clip.duration-1))]

    volumes_floats = [float(vol) for vol in volumes]
    volumes_strings = ["%.2f" % vol for vol in volumes_floats]

    volume_mean = np.mean(volumes)
    volume_std_dev = np.std(volumes)
    volume_min = np.min(volumes)
    volume_max = np.max(volumes)

    # Get a list of all of the differences between volumes (i.e., the derivative
        vector fo the volumes)
    volume_diffs = [cur_volume - volumes_floats[i - 1] for i, cur_volume in
        enumerate(volumes_floats)][1:]
    # Isolate sudden rises and falls
    print 'volume_diffs:', volume_diffs
    sudden_rise_count = sum([diff > volume_std_dev for diff in volume_diffs])
    sudden_fall_count = sum([diff < -volume_std_dev for diff in volume_diffs])

    print 'sudden_rise_count:', sudden_rise_count
    print 'sudden_fall_count:', sudden_fall_count

    # Normalize to audio length
    sudden_rise_count_per_cut = float(sudden_rise_count) / float(len(volumes))
    sudden_fall_count_per_cut = float(sudden_fall_count) / float(len(volumes))

    # Get rid of super-small mins
    epsilon = 0.001
    if volume_min < epsilon:
        volume_min = 0.0

    return float(volume_mean), float(volume_std_dev), float(volume_min), float(
        volume_max), sudden_rise_count_per_cut, sudden_fall_count_per_cut,
        volumes_strings

# Get video-related characteristics for the video
def analyze_video(filename):
    print 'Analyzing video for', filename

    # Get a camera input source
    cap = cv2.VideoCapture(filename)
```

```
fps = cap.get(cv2.CAP_PROP_FPS)

frameNo = 0

avg_colors = []

shot_transition_threshold = 100000
shot_transitions = [0.0]

dark_threshold = 5
consecutive_dark_frame_count = 0
# This will be a list of all of the "dark scenes," by frame count
dark_scene_list = []

# List for "detail scores" calculated from Canny edge detection
detail_scores = []

# List for average colors
avg_color_list = []

while(cap.isOpened()):
    frame = getFrame(cap)
    if frame is None:
        break

    height, width = frame.shape[:2]
    dimensional_ratio = float(width) / float(height)

    # Initialize frame accumulator, if necessary
    if frameNo == 0:
        frame_accumulator = np.zeros((height, width, 3), np.uint64)

    current_time = float(frameNo) / float(fps)

    # Check to make sure this isn't a fully black frame (if it is, averaging it
    # into the other colors will throw us off)
    if not np.all(np.less(frame, dark_threshold)):
        # # Get the average color for this frame
        # avg_color = cv2.mean(frame)[:3]
        # avg_colors.append(avg_color)

        # Accumulate this frame's pixels
        frame_accumulator = np.add(frame_accumulator, frame)

        # If this was the end of a "dark scene" (i.e., a dark transition),
        # then let's dump the current dark frame count into the dark scene
        # list and reset the count
        if consecutive_dark_frame_count > 0:
            dark_scene_list.append(consecutive_dark_frame_count)
            consecutive_dark_frame_count = 0

        # Debug prints for color debugging
        avg_color_this_frame = cv2.mean(frame)[:3]
        avg_color_list.append(avg_color_this_frame)
        # avg_intensity_this_frame = float(avg_color_this_frame[0] * 0.2989 +
            avg_color_this_frame[1] * 0.5870 + avg_color_this_frame[2] * 0.1140)
        # print frameNo, avg_intensity_this_frame
        # if avg_intensity_this_frame < 5.0:
```

```
            #     print frame

            # Find edges for detail score calculation
            edges = cv2.Canny(frame, 60, 200)
            detail_score = float(np.sum(edges)) / 255.0
            detail_scores.append(detail_score)

            # Edge detection debug code (comment out when not debugging edges)
            # cv2.imshow('Original Image', frame)
            # cv2.imshow('Edge-Detected Image', edges)
            # k = cv2.waitKey(1) & 0xFF
            # if k == 27:
            #     break
            # print 'Detail Score:', detail_score

        else:
            # Dark frame; skip on our averaging
            # print "dark frame"
            consecutive_dark_frame_count += 1

        # Calculate the color histogram for this frame and compare to the previous
            one
        # (if the chi-squared distance between the two histograms exceeds the shot
        # transition threshold, then we'll assume we've experienced a transition)
        current_frame_histogram = cv2.calcHist([frame], [0, 1, 2], None, [8, 8, 8],
            [0, 256, 0, 256, 0, 256])
        if frameNo != 0:
            chi_squared_distance = cv2.compareHist(current_frame_histogram,
                previous_frame_histogram, cv2.HISTCMP_CHISQR)
            # print frameNo, chi_squared_distance
            if chi_squared_distance > shot_transition_threshold:
                # Shot transition detected! We'll record the current time in seconds
                shot_transitions.append(current_time)

        previous_frame_histogram = current_frame_histogram

        frameNo += 1
        # print frameNo

    # If this was the end of a "dark scene" (i.e., a dark transition),
    # then let's dump the current dark frame count into the dark scene list
    if consecutive_dark_frame_count > 0:
        dark_scene_list.append(consecutive_dark_frame_count)

    # Now that we have an accumulated image, let's add together ALL of the pixels
    accumulated_color_sum = np.sum(np.sum(frame_accumulator, 1), 0)

    # print 'accumulated_color_sum:', accumulated_color_sum

    # Divide by the number of non-dark frames to get the average pixel sum for a
        frame
    avg_pixel_sum = np.true_divide(accumulated_color_sum, frameNo - sum(
        dark_scene_list))

    # print 'avg_pixel_sum:', avg_pixel_sum

    # Now, we just need to divide by the number of non-zero pixels (the pixels that
        are zero
```

```
# are probably letterbox pixels, and we should ignore them. (The np.all() below
    reduces
# the three RGB elements to just a single element, true when R, G, and B are all
    greater
# than the dark_threshold and false otherwise.)
num_pixels = np.sum(np.all(np.greater(frame_accumulator, dark_threshold), 2))

print 'num_pixels:', num_pixels

avg_color = np.true_divide(avg_pixel_sum, num_pixels).tolist()

# print 'avg_color:', avg_color

# Do a few calculations that tell us about color distribution, although they don
    't
# take into account the presence of a letterbox (too much of a pain in the butt
    this way)
avg_color_with_letterbox = np.mean(avg_color_list, 0)
stddev_color_with_letterbox = np.std(avg_color_list, 0)
min_color_with_letterbox = np.min(avg_color_list, 0)
max_color_with_letterbox = np.max(avg_color_list, 0)

# print "Mean of color without removing letterbox", avg_color_with_letterbox.
    tolist()
# print "Standard deviation of color without removing letterbox",
    stddev_color_with_letterbox.tolist()
# print "Min of color without removing letterbox", min_color_with_letterbox.
    tolist()
# print "Max of color without removing letterbox", max_color_with_letterbox.
    tolist()

# Output data related to "detail scores"
detail_score_mean = float(np.mean(detail_scores)) / num_pixels
detail_score_std_dev = float(np.std(detail_scores)) / num_pixels
detail_score_max = float(np.max(detail_scores)) / num_pixels
detail_score_min = float(np.min(detail_scores)) / num_pixels

# print 'detail_score_mean:', detail_score_mean
# print 'detail_score_std_dev:', detail_score_std_dev
# print 'detail_score_max:', detail_score_max
# print 'detail_score_min:', detail_score_min

# Convert to grayscale intensity using standard weights
avg_intensity = avg_color[0] * 0.2989 + avg_color[1] * 0.5870 + avg_color[2] *
    0.1140

# print 'avg_intensity:', avg_intensity

# Get various transition time-related data items
shot_lengths = [j-i for i, j in zip(shot_transitions[:-1], shot_transitions[1:])
    ]
mean_shot_length = float(np.mean(shot_lengths))
std_dev_shot_length = float(np.std(shot_lengths))
max_shot_length = float(np.max(shot_lengths))
min_shot_length = float(np.min(shot_lengths))
num_shots = len(shot_lengths)
```

```
    # Calculate data related to consecutive dark frames (pitch black transitions, a.
        k.a. dark scenes)
    # print "dark scene data:", dark_scene_list
    if len(dark_scene_list) > 0:
        dark_scene_mean_length = float(np.mean(dark_scene_list))
        dark_scene_length_std_dev = float(np.std(dark_scene_list))
        dark_scene_length_max = float(np.max(dark_scene_list))
        dark_scene_length_min = float(np.min(dark_scene_list))
    else:
        dark_scene_mean_length = 0.0
        dark_scene_length_std_dev = 0.0
        dark_scene_length_max = 0.0
        dark_scene_length_min = 0.0
    dark_scene_count = len(dark_scene_list)
    dark_scene_percentage = float(dark_scene_count) / float(frameNo)

    # print 'dark_scene_mean_length:', dark_scene_mean_length
    # print 'dark_scene_length_std_dev:', dark_scene_length_std_dev
    # print 'dark_scene_length_max:', dark_scene_length_max
    # print 'dark_scene_length_min:', dark_scene_length_min
    # print 'dark_scene_count:', dark_scene_count
    # print 'dark_scene_percentage:', dark_scene_percentage

    return frameNo, current_time, avg_intensity, avg_color, mean_shot_length,
        std_dev_shot_length, max_shot_length, min_shot_length, num_shots,
        stddev_color_with_letterbox, detail_score_mean, detail_score_std_dev,
        detail_score_max, detail_score_min, dark_scene_mean_length,
        dark_scene_length_std_dev, dark_scene_length_max, dark_scene_length_min,
        dark_scene_count, dark_scene_percentage

def main():
    # Create video analysis (or audio analysis) results file, in case it doesn't
        already exist
    open(analysis_results_file, 'a').close()

    # Go through all directories
    for dirname, dirnames, filenames in os.walk(video_dir):
        # print dirname
        # print dirnames
        # print filenames
        # print '---'

            # Only look at non-empty directories
            if len(filenames) > 0:
                for filename in filenames:
                    # Parse out the movie name from the directory name
                    movie_title = dirname.split('\\')[-1]

                    if is_video_already_analyzed(movie_title):
                        print "%s already analyzed; skipping" % movie_title
                        continue
                    else:
                        if video_analysis_mode:
                            num_frames, total_time, avg_intensity, avg_color,
                                mean_shot_length, std_dev_shot_length,
                                max_shot_length, min_shot_length, num_shots,
                                stddev_color_with_letterbox, detail_score_mean,
                                detail_score_std_dev, detail_score_max,
```

```
            detail_score_min, dark_scene_mean_length,
            dark_scene_length_std_dev, dark_scene_length_max,
            dark_scene_length_min, dark_scene_count,
            dark_scene_percentage = analyze_video(os.path.join(
            dirname, filename))
    else:
        mean_volume, std_dev_volume, min_volume, max_volume,
            sudden_rise_count_per_cut, sudden_fall_count_per_cut
            , volumes_strings = analyze_sound(os.path.join(
            dirname, filename))

print 'Movie title:', movie_title

if video_analysis_mode:
    print 'Number of frames:', num_frames
    print 'Total time (s):', total_time
    print 'Average Pixel Intensity', avg_intensity
    print 'Average Pixel Color:', avg_color
    print 'Average Shot Length (s):', mean_shot_length
    print 'Shot Length Standard Deviation (s):',
        std_dev_shot_length
    print 'Shot Length Maximum (s):', max_shot_length
    print 'Shot Length Minimum (s):', min_shot_length
    print 'Number of Shots:', num_shots
    print 'Standard deviation of color (with letterbox):',
        stddev_color_with_letterbox
    print 'Mean detail score:', detail_score_mean
    print 'Detail score standard deviation:',
        detail_score_std_dev
    print 'Maximum detail score:', detail_score_max
    print 'Minimum detail score:', detail_score_min
    print 'Mean length of transitional black scenes:',
        dark_scene_mean_length
    print 'Standard deviation of length of transitional black
        scenes:', dark_scene_length_std_dev
    print 'Max length of transitional black scenes:',
        dark_scene_length_max
    print 'Min length of transitional black scenes:',
        dark_scene_length_min
    print 'Total count of transitional black scenes:',
        dark_scene_count
    print 'Percentage of trailer occupied by transitional black
        scene frames:', dark_scene_percentage
else:
    print 'Mean Volume:', mean_volume
    print 'Volume Standard Deviation:', std_dev_volume
    print 'Minimum Volume', min_volume
    print 'Maximum Volume', max_volume
    print 'Number of Sudden Volume Rises Per Second of Audio',
        sudden_rise_count_per_cut
    print 'Number of Sudden Volume Falls Per Second of Audio',
        sudden_fall_count_per_cut
    print 'Volumes Strings', volumes_strings

# Open file in which to log all of this data
with open(analysis_results_file, 'ab') as outfile:
    # Output this data
    outfile.write('movie_title: ' + movie_title + '\n')
```

```
            if video_analysis_mode:
                # Output video analysis metrics
                outfile.write('num_frames: ' + str(num_frames) + '\n')
                outfile.write('total_time: ' + str(total_time) + '\n')
                outfile.write('avg_intensity: ' + str(avg_intensity) + '
                    \n')
                outfile.write('avg_color: ' + str(avg_color) + '\n')
                outfile.write('mean_shot_length: ' + str(
                    mean_shot_length) + '\n')
                outfile.write('std_dev_shot_length: ' + str(
                    std_dev_shot_length) + '\n')
                outfile.write('max_shot_length: ' + str(max_shot_length)
                     + '\n')
                outfile.write('min_shot_length: ' + str(min_shot_length)
                     + '\n')
                outfile.write('num_shots: ' + str(num_shots) + '\n')
                outfile.write('stddev_color_with_letterbox: ' + str(
                    stddev_color_with_letterbox) + '\n')
                outfile.write('detail_score_mean: ' + str(
                    detail_score_mean) + '\n')
                outfile.write('detail_score_std_dev: ' + str(
                    detail_score_std_dev) + '\n')
                outfile.write('detail_score_max: ' + str(
                    detail_score_max) + '\n')
                outfile.write('detail_score_min: ' + str(
                    detail_score_min) + '\n')
                outfile.write('dark_scene_mean_length: ' + str(
                    dark_scene_mean_length) + '\n')
                outfile.write('dark_scene_length_std_dev: ' + str(
                    dark_scene_length_std_dev) + '\n')
                outfile.write('dark_scene_length_max: ' + str(
                    dark_scene_length_max) + '\n')
                outfile.write('dark_scene_length_min: ' + str(
                    dark_scene_length_min) + '\n')
                outfile.write('dark_scene_count: ' + str(
                    dark_scene_count) + '\n')
                outfile.write('dark_scene_percentage: ' + str(
                    dark_scene_percentage) + '\n\n')
            else:
                # Output audio analysis metrics
                outfile.write('mean_volume: ' + str(mean_volume) + '\n')
                outfile.write('std_dev_volume: ' + str(std_dev_volume) +
                     '\n')
                outfile.write('min_volume: ' + str(min_volume) + '\n')
                outfile.write('max_volume: ' + str(max_volume) + '\n')
                outfile.write('sudden_rise_count_per_cut: ' + str(
                    sudden_rise_count_per_cut) + '\n')
                outfile.write('sudden_fall_count_per_cut: ' + str(
                    sudden_fall_count_per_cut) + '\n\n')


if __name__ == '__main__':
    main()
```

H. expand_genres.py

```
# -*- coding: utf-8 -*-
```

```
"""
Created on Thu Feb 25 14:30:50 2016

@author: fuini
"""


import csv
import re

csv_filename = "trailer_data.csv"

my_dict = {}

# list of my genres
no_genre_movie_list = []
genre_list = []
genre_count = {}

#findin the genres and the movies without genres
with open(csv_filename, 'r') as csv_file:
    reader = csv.DictReader(csv_file)
    for i, row in enumerate(reader):
        if row["genre"]:
            # build genre list being all regexy like Nat
            genres = re.findall('(\w+)', row["genre"])
            for genre in genres:
                if genre in genre_list:
                    genre_count[genre] += 1
                else:
                    genre_list.append(genre)
                    genre_count[genre] = 1

            # holy shit that worked.

        else:  #note who doesn't have a genre
            no_genre_movie_list.append(row["title"])

print "Number of movies with no genre: " + str(len(no_genre_movie_list))
# 68 movies have no genre.
print "Number of unique genres: " + str(len(genre_list))
print genre_list
# 25 genres
print "Each genre and number of times present in data: "
print genre_count
print "We should seriously consider removing genres that have only a few
    representatives, as we don't have the statistics to say anything meaningful."



#Read in and save CSV into my_dict, while building new rows into my_dict
with open(csv_filename, 'r') as csv_file:
    reader = csv.DictReader(csv_file)
    for i, row in enumerate(reader):
        my_dict[row["title"]] = {}
        for key in row.keys():
            my_dict[row["title"]][key] = row[key]
            #collect genres of movie
```

```
            genres_of_movie = re.findall('(\w+)', row["genre"])
            #loop through all genres, append 0 or 1 for each genre
            for genre in genre_list:
                if genre in genres_of_movie:
                    my_dict[row["title"]][genre] = 1
                else:
                    my_dict[row["title"]][genre] = 0
        # check genre business
        # loop over my list of genres
print my_dict["Step-Up-All-In"]


#now write out my dict to a csv
final_csv_filename = "trailer_data_expanded_genre.csv"
with open(final_csv_filename, 'wb') as csv_file:
    fieldnames = ['title', 'num_frames', 'total_time', 'avg_intensity', 'avg_color_r
        ', 'avg_color_g', 'avg_color_b', 'avg_shot_length', 'num_shots', '
        original_format_title', 'movie_list_webpage_url', 'genre', 'release_date', '
        imdb_url', 'mean_volume', 'std_dev_volume', 'min_volume', 'max_volume', '
        Drama', 'Horror', 'Thriller', 'Action', 'Comedy', 'Crime', 'Mystery', 'Sport
        ', 'Romance', 'Biography', 'History', 'Animation', 'Adventure', 'Family', '
        Documentary', 'Fantasy', 'Music', 'Western', 'Musical', 'Sports', '
        Supernatural', 'War', 'News', 'Animaton', 'Short']
    writer = csv.DictWriter(csv_file, fieldnames=fieldnames)

    writer.writeheader()

    for key in my_dict.keys():
        writer.writerow(my_dict[key])
```

I. analyze_main.m

```
%% Import data from text file.
% Script for importing data from the following text file:
%
%    C:\Users\johnf_000\Dropbox\MATLAB\Assignments\Final Project\Analysis\
    more_trailer_dataNOCOMMA_w_octave_standardized.csv
%
% To extend the code to different selected data or a different text file,
% generate a function instead of a script.

% Auto-generated by MATLAB on 2016/03/08 21:20:19

%% Initialize variables.
filename = 'C:\Users\johnf_000\Dropbox\MATLAB\Assignments\Final_Project\Analysis\
    more_trailer_dataNOCOMMA_w_octave_standardized.csv';
delimiter = ',';
startRow = 2;

%% Format string for each line of text:
%   column2: double (%f)
%       column3: double (%f)
%   column4: double (%f)
%       column5: double (%f)
%   column6: double (%f)
%       column7: double (%f)
%   column8: double (%f)
%       column9: double (%f)
%   column10: double (%f)
```

```
%       column11: double (%f)
%   column12: double (%f)
%       column13: double (%f)
%   column14: double (%f)
%       column15: double (%f)
%   column16: double (%f)
%       column17: double (%f)
%   column18: double (%f)
%       column19: double (%f)
%   column20: double (%f)
%       column21: double (%f)
%   column22: double (%f)
%       column23: double (%f)
%   column24: double (%f)
%       column25: double (%f)
%   column26: double (%f)
%       column27: double (%f)
%   column28: double (%f)
%       column29: double (%f)
%   column30: double (%f)
%       column31: double (%f)
%   column32: double (%f)
%       column33: double (%f)
%   column34: double (%f)
%       column35: double (%f)
%   column36: double (%f)
%       column37: double (%f)
%   column38: double (%f)
%       column39: double (%f)
%   column40: double (%f)
%       column41: double (%f)
%   column42: double (%f)
%       column43: double (%f)
%   column44: double (%f)
%       column45: double (%f)
%   column46: double (%f)
%       column47: double (%f)
%   column48: double (%f)
%       column49: double (%f)
%   column50: double (%f)
%       column51: double (%f)
%   column52: double (%f)
%       column53: double (%f)
%   column54: double (%f)
%       column55: double (%f)
%   column56: double (%f)
%       column57: double (%f)
%   column58: double (%f)
%       column59: double (%f)
%   column60: double (%f)
%       column61: double (%f)
%   column62: double (%f)
%       column63: double (%f)
%   column64: double (%f)
%       column65: double (%f)
%   column66: double (%f)
%       column67: double (%f)
% For more information, see the TEXTSCAN documentation.
```

```
formatSpec = '%*s%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%
    f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f %[^\n\r]';

%% Open the text file.
fileID = fopen(filename,'r');

%% Read columns of data according to format string.
% This call is based on the structure of the file used to generate this
% code. If an error occurs for a different file, try regenerating the code
% from the Import Tool.
dataArray = textscan(fileID, formatSpec, 'Delimiter', delimiter, 'HeaderLines' ,
    startRow-1, 'ReturnOnError', false);

%% Close the text file.
fclose(fileID);

%% Post processing for unimportable data.
% No unimportable data rules were applied during the import, so no post
% processing code is included. To generate code which works for
% unimportable data, select unimportable cells in a file and regenerate the
% script.

%% Allocate imported array to column variable names
num_frames = dataArray{:, 1};
total_time = dataArray{:, 2};
avg_intensity = dataArray{:, 3};
avg_color_r = dataArray{:, 4};
avg_color_g = dataArray{:, 5};
avg_color_b = dataArray{:, 6};
mean_shot_length = dataArray{:, 7};
std_dev_shot_length = dataArray{:, 8};
max_shot_length = dataArray{:, 9};
min_shot_length = dataArray{:, 10};
num_shots = dataArray{:, 11};
stddev_color_with_letterbox_r = dataArray{:, 12};
stddev_color_with_letterbox_g = dataArray{:, 13};
stddev_color_with_letterbox_b = dataArray{:, 14};
detail_score_mean = dataArray{:, 15};
detail_score_std_dev = dataArray{:, 16};
detail_score_max = dataArray{:, 17};
detail_score_min = dataArray{:, 18};
dark_scene_mean_length = dataArray{:, 19};
dark_scene_length_std_dev = dataArray{:, 20};
dark_scene_length_max = dataArray{:, 21};
dark_scene_length_min = dataArray{:, 22};
dark_scene_count = dataArray{:, 23};
dark_scene_percentage = dataArray{:, 24};
mean_volume = dataArray{:, 25};
std_dev_volume = dataArray{:, 26};
min_volume = dataArray{:, 27};
max_volume = dataArray{:, 28};
sudden_rise_count_per_cut = dataArray{:, 29};
sudden_fall_count_per_cut = dataArray{:, 30};
octave1 = dataArray{:, 31};
octave2 = dataArray{:, 32};
octave3 = dataArray{:, 33};
octave4 = dataArray{:, 34};
octave5 = dataArray{:, 35};
```

```
octave6 = dataArray{:, 36};
octave7 = dataArray{:, 37};
octave8 = dataArray{:, 38};
octave9 = dataArray{:, 39};
octave10 = dataArray{:, 40};
octave11 = dataArray{:, 41};
Drama = dataArray{:, 42};
Horror = dataArray{:, 43};
Thriller = dataArray{:, 44};
Action = dataArray{:, 45};
Comedy = dataArray{:, 46};
Crime = dataArray{:, 47};
Mystery = dataArray{:, 48};
Sport = dataArray{:, 49};
Romance = dataArray{:, 50};
Biography = dataArray{:, 51};
History = dataArray{:, 52};
Animation = dataArray{:, 53};
Adventure = dataArray{:, 54};
Family = dataArray{:, 55};
Documentary = dataArray{:, 56};
Fantasy = dataArray{:, 57};
Music = dataArray{:, 58};
Western = dataArray{:, 59};
Musical = dataArray{:, 60};
Sports = dataArray{:, 61};
Supernatural = dataArray{:, 62};
War = dataArray{:, 63};
News = dataArray{:, 64};
Animaton = dataArray{:, 65};
Short = dataArray{:, 66};


%% Clear temporary variables
clearvars filename delimiter startRow formatSpec fileID dataArray ans;

%% Data Analysis

%each row is a trailer, and we have 40 features
master_data = [total_time avg_intensity avg_color_r avg_color_g avg_color_b...
    mean_shot_length std_dev_shot_length max_shot_length min_shot_length...
    num_shots stddev_color_with_letterbox_r stddev_color_with_letterbox_g...
    stddev_color_with_letterbox_b detail_score_mean detail_score_std_dev...
    detail_score_max detail_score_min dark_scene_mean_length...
    dark_scene_length_std_dev dark_scene_length_max     dark_scene_length_min...
    dark_scene_count dark_scene_percentage mean_volume...
    std_dev_volume min_volume max_volume sudden_rise_count_per_cut...
    sudden_fall_count_per_cut octave1 octave2 octave3 octave4 octave5 octave6...
    octave7 octave8 octave9 octave10 octave11];




[num_movies, num_features] = size(master_data);


%% Testing module
```

```
clearvars accuracy
accuracy = [];
num_trials = 40; % number of cross validation trials
accuracy(1) = check_genre_predictions(master_data, Drama, 0.8, num_trials);
accuracy(2) = check_genre_predictions(master_data, Comedy, 0.8, num_trials);
accuracy(3) = check_genre_predictions(master_data, Thriller, 0.8, num_trials);
accuracy(4) = check_genre_predictions(master_data, Action, 0.8, num_trials);
accuracy(5) = check_genre_predictions(master_data, Horror, 0.8, num_trials);
accuracy(6) = check_genre_predictions(master_data, Crime, 0.8, num_trials);
accuracy(7) = check_genre_predictions(master_data, Romance, 0.8, num_trials);
accuracy(8) = check_genre_predictions(master_data, Adventure, 0.8, num_trials);
accuracy(9) = check_genre_predictions(master_data, Biography, 0.8, num_trials);
accuracy(10) = check_genre_predictions(master_data, Documentary, 0.8, num_trials);

%% Plot
figure(1)
clearvars success
success = [];
for j = 1:length(accuracy)
    success(j) = 1 - accuracy(j);
end

plot(success, 'ko', 'LineWidth',[2.0]), axis([1 10 0 1])
title('Decision Tree Prediction Success for Most Populated Genres')
xlabel('Drama, Comedy, Thriller, Action, Horror, Crime, Romance, Adventure, Biography, Documentary')
%% SVD for Dimensional Reduction

[m,n] = size(master_data);
mn=mean(master_data,2);
master_data = master_data - repmat(mn,1,n);

[u,s,v] = svd(master_data'/(sqrt(n-1)));
lambda = diag(s).^2;

%% plot singular values
figure(2)
subplot(1,2,1), plot(diag(s)/sum(diag(s)), 'ko', 'LineWidth',[1.5])
title('Singular Values normalized'), xlabel('Principle Mode Number'), ylabel('')
subplot(1,2,2), semilogy(lambda, 'ko', 'LineWidth',[1.5])
title('Singular Values on logplot'), xlabel('Principle Mode Number'), ylabel('')

%% Building Reconstructions


ff = u*s*v'; %full data
ff1 = u(:,1)*s(1,1)*v(:,1)'; %one mode
ff2 = u(:,1:2)*s(1:2,1:2)*v(:,1:2)'; %two mode
ff4 = u(:,1:4)*s(1:4,1:4)*v(:,1:4)'; %four mode

figure(3)
plot(1:length(u(:,1)), abs(u(:,1)), 'ko', ...
    1:length(u(:,2)), abs(u(:,2)), 'rx', ...
    1:length(u(:,3)), abs(u(:,3)), 'bv', ...
    1:length(u(:,4)), abs(u(:,4)), 'g+', 'LineWidth', [2.0]); legend('Mode One', '
        Mode Two', 'Mode Three', 'Mode Four', 'Location', 'best');
title('Components of first four principle modes'); xlabel('Features')
```

```
%% Testing reconstruction
clearvars accuracyOne
accuracyOne = [];
data = ff1';
num_trials = 40; % number of cross validation trials
accuracyOne(1) = check_genre_predictions(data, Drama, 0.8, num_trials);
accuracyOne(2) = check_genre_predictions(data, Comedy, 0.8, num_trials);
accuracyOne(3) = check_genre_predictions(data, Thriller, 0.8, num_trials);
accuracyOne(4) = check_genre_predictions(data, Action, 0.8, num_trials);
accuracyOne(5) = check_genre_predictions(data, Horror, 0.8, num_trials);
accuracyOne(6) = check_genre_predictions(data, Crime, 0.8, num_trials);
accuracyOne(7) = check_genre_predictions(data, Romance, 0.8, num_trials);
accuracyOne(8) = check_genre_predictions(data, Adventure, 0.8, num_trials);
accuracyOne(9) = check_genre_predictions(data, Biography, 0.8, num_trials);
accuracyOne(10) = check_genre_predictions(data, Documentary, 0.8, num_trials);

%% Another Reconstruction test
clearvars accuracyTwo
accuracyTwo = [];
data = ff4';
num_trials = 40; % number of cross validation trials
accuracyTwo(1) = check_genre_predictions(data, Drama, 0.8, num_trials);
accuracyTwo(2) = check_genre_predictions(data, Comedy, 0.8, num_trials);
accuracyTwo(3) = check_genre_predictions(data, Thriller, 0.8, num_trials);
accuracyTwo(4) = check_genre_predictions(data, Action, 0.8, num_trials);
accuracyTwo(5) = check_genre_predictions(data, Horror, 0.8, num_trials);
accuracyTwo(6) = check_genre_predictions(data, Crime, 0.8, num_trials);
accuracyTwo(7) = check_genre_predictions(data, Romance, 0.8, num_trials);
accuracyTwo(8) = check_genre_predictions(data, Adventure, 0.8, num_trials);
accuracyTwo(9) = check_genre_predictions(data, Biography, 0.8, num_trials);
accuracyTwo(10) = check_genre_predictions(data, Documentary, 0.8, num_trials);
%% Plot
figure(2)
clearvars success
success = [];
for j = 1:length(accuracy)
    success(j) = 1 - accuracy(j);
end

clearvars successOne
successOne = [];
for j = 1:length(accuracyOne)
    successOne(j) = 1 - accuracyOne(j);
end
clearvars successTwo
successTwo = [];
for j = 1:length(accuracyTwo)
    successTwo(j) = 1 - accuracyTwo(j);
end

plot(1:10, success, 'ko',1:10, successOne, 'ro', 1:10, successTwo, 'bo', 'LineWidth'
    ,[2.0])
axis([1 10 .5 1]), legend('Full Data', 'One-Mode', 'Four-Mode', 'Location', 'best');
title('Classification Tree Success from Reconstructed Data')
xlabel('Drama, Comedy, Thriller, Action, Horror, Crime, Romance, Adventure,
    Biography, Documentary')
```

```
%% SVM testor (remove and create function)

SVMModel = fitcsvm(master_data, Documentary, 'Standardize', true)
CVSVMModel = crossval(SVMModel)
kfoldLoss(CVSVMModel)

%% Predictor

[label, score] = predict(SVMModel, master_data(10:206,:));
label - Documentary(10:206)

%% Accuracy SVM
% (checking using kfoldLoss, could potentially double check error using
% manual cross-validation
clearvars accuracySVM
accuracySVM = [];
SVMModel = fitcsvm(master_data, Drama, 'Standardize', true);
CVSVMModel = crossval(SVMModel);
accuracySVM(1) = kfoldLoss(CVSVMModel);

SVMModel = fitcsvm(master_data, Comedy, 'Standardize', true);
CVSVMModel = crossval(SVMModel);
accuracySVM(2) = kfoldLoss(CVSVMModel);

SVMModel = fitcsvm(master_data, Thriller, 'Standardize', true);
CVSVMModel = crossval(SVMModel);
accuracySVM(3) = kfoldLoss(CVSVMModel);

SVMModel = fitcsvm(master_data, Action, 'Standardize', true);
CVSVMModel = crossval(SVMModel);
accuracySVM(4) = kfoldLoss(CVSVMModel);

SVMModel = fitcsvm(master_data, Horror, 'Standardize', true);
CVSVMModel = crossval(SVMModel);
accuracySVM(5) = kfoldLoss(CVSVMModel);

SVMModel = fitcsvm(master_data, Crime, 'Standardize', true);
CVSVMModel = crossval(SVMModel);
accuracySVM(6) = kfoldLoss(CVSVMModel);

SVMModel = fitcsvm(master_data, Romance, 'Standardize', true);
CVSVMModel = crossval(SVMModel);
accuracySVM(7) = kfoldLoss(CVSVMModel);


SVMModel = fitcsvm(master_data, Adventure, 'Standardize', true);
CVSVMModel = crossval(SVMModel);
accuracySVM(8) = kfoldLoss(CVSVMModel);

SVMModel = fitcsvm(master_data, Biography, 'Standardize', true);
CVSVMModel = crossval(SVMModel);
accuracySVM(9) = kfoldLoss(CVSVMModel);


SVMModel = fitcsvm(master_data, Documentary, 'Standardize', true);
CVSVMModel = crossval(SVMModel);
accuracySVM(10) = kfoldLoss(CVSVMModel);
```

```
%% Plot SVM

figure(3)
clearvars success
successSVM = [];
for j = 1:length(accuracySVM)
    successSVM(j) = 1 - accuracySVM(j);
end

plot(successSVM, 'ko', 'LineWidth',[2.0]), axis([1 10 0 1])
title('SVM Prediction Success for Most Populated Genres')
xlabel('Drama, Comedy, Thriller, Action, Horror, Crime, Romance, Adventure,
    Biography, Documentary')

%% SVM vs Tree
figure(3)
clearvars successSVM
successSVM = [];
for j = 1:length(accuracySVM)
    successSVM(j) = 1 - accuracySVM(j);
end

clearvars success
success = [];
for j = 1:length(accuracy)
    success(j) = 1 - accuracy(j);
end



plot(1:10, successSVM, 'ko', 1:10, success, 'go', 'LineWidth',[2.0]), axis([1 10 0.5
    1])
title('Prediction Success for Most Populated Genres')
xlabel('Drama, Comedy, Thriller, Action, Horror, Crime, Romance, Adventure,
    Biography, Documentary')
legend( 'Support Vector Machine','Classification Tree', 'Location', 'best')
```

J. check_genre_predictions.m

```
function A = check_genre_predictions(data, genre, fraction_to_train, num_trials)

failed_fractions = [];

for j = 1:num_trials
    [num_movies, num_features] = size([data]);

    rand_order = randperm(num_movies); % random ordering

    train_to = floor(fraction_to_train*num_movies);

    train_data = data(rand_order(1:train_to),:);
    test_data = data(rand_order(train_to+1:end),:);
    train_labels =  genre(rand_order(1:train_to));
    test_labels = genre(rand_order(train_to+1:end));

    tree = fitctree(train_data, train_labels); % train on the first 80% of movies

    prediction = predict(tree,test_data);
    failed_fraction = sum(abs(test_labels-prediction))/(num_movies-train_to);
```

```
    failed_fractions(j) = failed_fraction;

end
% view(tree)

A = sum(failed_fractions)/length(failed_fractions);
```