Multimedia Feature Generation of Movie Trailers for Genre Prediction

John Fuini, Nathaniel Guy, and Yong Han Noel Kim
University of Washington, Seattle WA

***Abstract*— This paper presents a collection of techniques for the generation of visual and audio features based on film trailer data, and then shows a viable machine learning solution for building a classifier using these features which can identify the genre of a previously unseen film trailer. Our approach sought to capture, using computer vision and audio analysis techniques, the essential data within the trailer which is more characteristically informative as to the film's genre. These features included average intensity and color, number and length of distinct shots, amount of detail, amount of consecutive darkness, average volume, sudden changes in sound, and frequency distribution of sound. Using these features, along with genre metadata for each trailer, we trained both a Binary Decision Tree and a Support Vector Machine model to identify the genres. We applied extensive cross-validation and demonstrated classification accuracy greater than 65% for the top ten most common genres, and greatrer than 90% for five of the top ten most common genres. We also performed a Singular Value Decomposition (SVD) to identify the dominant modes of the feature set, and showed that the success rate of a classifier built from a one-mode reconstruction of the data only reduced our accuracy by, at most, 6%. We envision that our technique can be improved and applied to a variety of problems, and may be capable of competing with human accuracy on a similar problem.**

I. INTRODUCTION

Movie trailers are one of the most effective advertising tools for the film industry. They deliver relevant information such as background, cast, theme, plot and more, in a limited amount of time. As such, trailers can be considered a subset of a movie which contains its principal components. With this idea in mind, we developed an algorithm which classifies movie trailers by their genre. This is a familiar process to all movie-viewers: movie viewers are generally able to tell if a certain movie is a comedy film, action film, documentary, etc. within the first minute of watching a trailer based on myriad cinematic features within it. Viewers have developed this cognitive ability by watching countless movies of various genres over time, and have subconsciously learned to identify the cinematic features associated with certain genres. Our algorithm is an adaptation of this process using machine learning. This report describes our process of identifying cinematic features from a large set of trailers of known genre, training a machine learning algorithm to develop classification criteria based on these features and genre metadata, and testing the generated classification criteria on a set of trailers to assess the effectiveness of our approach.

Nathaniel Guy and Yong Han Noel Kim are Masters students in the University of Washington Department of Aeronautical and Astronautical Engineering, and can be reached at natguy@cs.washington.edu and kimber.noel@outlook.com, respectively. John Fuini is a PhD student in the University of Washington Department of Physics, and can be reached at fuini@uw.edu.

A. Sample Data Set

For the bulk of our classifier training and testing, we used trailer data from roughly 1,000 major motion pictures released within the last decade. These trailers were all high-resolution (the majority were 720p), with a typical framerate of 24 FPS. The trailers were downloaded from online sources using custom-built scripts, along with movie metadata (such as genre), which was scraped from wwww.movie-list.com.

II. RELATED WORK

Zeeshan Rasheed et al., in their *On the Use of Computable Features for Film Classification*[1], developed an algorithm for film classification based on film previews. They limited themselves to visual features only, such as average shot length, color variance, motion content and lighting keys, and constrained their classification to four genres: comedy, action, drama and horror. In contrast, our work aims to create an algorithm that can classify a larger number of genres, using more features derived from both video and audio features, and with greater robustness than the technique in [1].

III. COMPONENT ARCHITECTURE

A. Feature Generation via Video Processing

We implemented a number of computer vision techniques to calculate features from each trailer's video data. The majority of our video processing was utilized the OpenCV "cv2" module in the Python programming language [2].

1) Total Time/Number of Frames: OpenCV allows the processing of video data on a per-frame basis. Individual frames were counted, in order to get a total run-time of any given trailer in terms of frame count. OpenCV can provide the frames per second (FPS) for a given trailer as well, and this allowed us to calculate the run-time of trailers in seconds.

2) Average Intensity: Average grayscale intensity, across all pixels and all frames, was calculated using standard RGB weights for grayscale reduction:

$$Average\,intensity = 0.2989R + 0.5870G + 0.1140B$$

Other statistical metrics, such as intensity standard deviation, min, and max, were calculated as additional features. Note that certain regions of many trailers constituted a black letterbox, and could be excluded from this calculation after the determination that pixels at certain coordinate positions remain black throughout the entire duration of a trailer.

3) R, G and B Components: R, G and B components denote the proportion of red, green and blue colors for a given pixel. We calculated the average intensities along each of these color channels, across pixels and across all frames. (In order to minimize runtime memory requirements and simplify code structure, we elected to include the letter-box regions in this calculation for some of the statistical measures, such as color channel standard deviations and minimum/maximum intensities.)

4) Number of Shots: We detected a shot transition by examining and comparing color histograms of adjacent frames. When the chi-squared distance between two histograms exceeds a predetermined shot transition threshold, we determine that there was a shot transition between the two frames. The predetermined threshold was made by hand by watching trailers and tweaking our shot detection until it performed satisfactorily. One disadvantage to this method is that algorithm cannot differentiate between fast camera movement and complete change of shots, since with fast camera movement, the distribution of colors within the frame can vary as much as with an actual change of shot, especially with large moving objects. See Fig. 1 for an example of frames across a shot transition and their histograms.

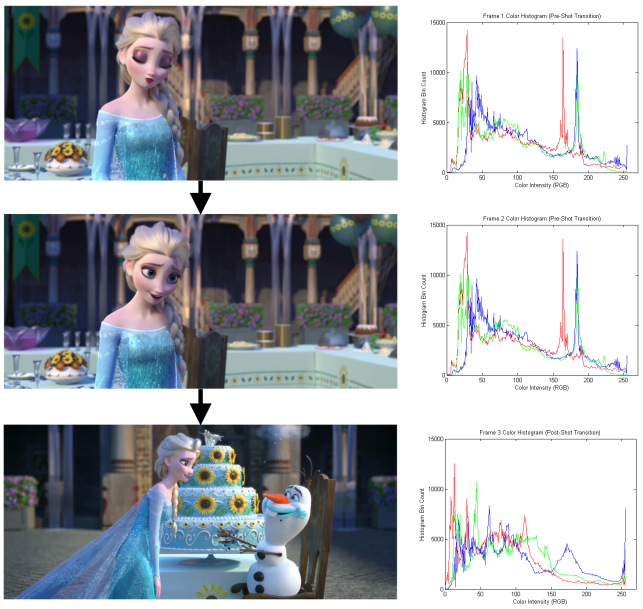

Fig. 1: Three example video frames are shown on the left, with their color histograms on the right. The first two frames are similar, and are part of the same shot; note their similar histograms. The third frame, only a fraction of a second later, has a starkly different histogram, and would thus be detected as a new shot.

5) Shot Length: Once the time-stamps of shot transitions were recognized, we were also able to calculate metrics based on the length of shots: mean, standard deviation, and minimum/maximum shot lengths. The mean shot length, for each trailer, varied greatly on our sample set, from 0.2 seconds to 3 seconds.

6) Detail Score: We defined a measure of the amount of complex detail in a trailer through the use of the Canny edge detection filter [3]. By applying the Canny algorithm to a given frame, we can calculate a binary mask wherein all of the "edge pixels" in that frame have a value of 1. Then, by summing all of pixels in the frame, and summing all of the frames into the trailer, we can get a total "detail score" for that trailer. That score can then be scaled by the number of frames to normalize and find the average detail per frame. We calculated detail score features (including mean value, standard deviation, and minimum/maximum values) in this way. See Fig. 2 for an example of edge-detected images from which detail scores were calculated.

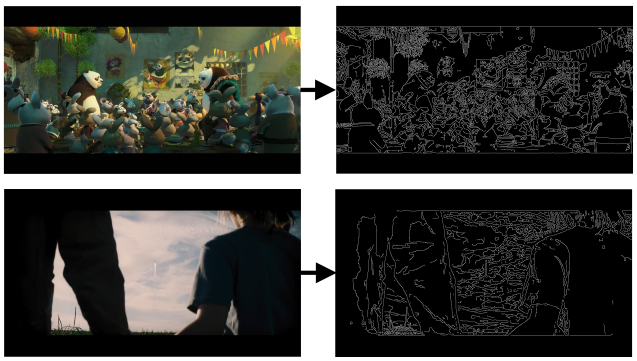

Fig. 2: Two example video frames are shown on the left, with their Canny edge-detected versions on the right. By counting the white (edge) pixels in the images on the right, a "detail score" can be calculated, as a rough metric of the detail complexity of the frame.

7) Dark Scenes: We defined the notion of a "dark scene" as a period of consecutive frames with very low average values (i.e., black transitions). Their lengths were recorded, from which mean length, standard deviation of length, and minimum/maximum lengths were calculated. The percentage of dark scene frames in each trailer was also calculated.

B. Feature Generation via Audio Processing

From the trailer videos, we isolated the audio component using a sampling frequency of 44.1 kHz. A series of analyses was performed on this audio data in order to extract features.

1) Volume: Mean: Mean volume for each trailer was calculated by averaging the amplitudes of sound waves over the entire duration of the trailer. One motivation behind extracting this feature was that a trailer saturated with loud noises would have larger value of mean volume than trailers with relatively calm sounds. Also, typically trailers with loud noises–explosions, jet noise, shouting, etc.–are associated with genres such as action, thriller, and adventure. On the other hand, trailers with calm audio, and even some quietness, may be associated with genres such as drama, history, and family.

Standard Deviation: all trailers were sourced from a single film site, but there was no guarantee that their audio was all equalized to the similar degree, especially because many

were produced by different companies. Thus, a higher mean volume could simply indicate a trailer that's louder in general due to different equalization characteristics, rather than an abundance of loud sound events. In order to get a sense of the variation of volume in each trailer, we calculated standard deviation of the sound wave amplitude over the entire trailer as well.

Minimum and Maximum: Minimum and maximum volume for each trailer were calculated based on the waveform data. On a scale from 0.0 to 1.0, most trailers had a minimum volume near 0.0, while some had marginally higher minimum values, such as 0.03. Maximum volume generally fell into the range of 0.1 to 0.5.

2) Sudden Rise/Fall of Volume: We defined the concept of a sudden rise and sudden fall of volume. Respectively, these represent an increase and a decrease of volume within a small time period, possessing a magnitude larger than the standard deviation of the volume across the whole trailer. Identifying the number of these events within a trailer's audio allowed us to study its audio dynamics, as we intuited that sudden increases of volume might be common during trailers that sought to startle viewers (such as those horror or action films).

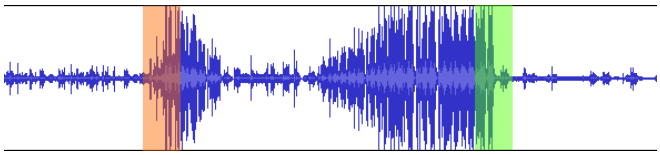

Fig. 3: An audio sample is shown, with a sudden rise and a sudden fall highlighted in orange and green, respectively. Note that the interior fall and rise are not captured as sudden changes, due to their more gradual change.

3) Percentage of Sound Corresponding to Different Octave Bands: For this feature, the waveform of each trailer was transformed to the frequency domain using the Fast Fourier transform (FFT). Its frequencies were divided into eleven bands in the audible range, commonly defined as octave bands (11Hz $\sim$ 22720Hz). The magnitudes of the frequency components in each band were summed together, and normalized so that the sum of magnitudes of all octave bands would be 1. The resulting binned magnitudes represented the composition of sounds of each trailer with respect to these eleven octave bands.

C. Use of Features in Machine Learning Algorithm

All extracted features from each trailer were compiled into a single comma-separated variable (CSV) spreadsheet. In addition to our generated features, the spreadsheet contained the genre labels for each trailer as well. Movies were not limited to one genre. For instance, there were movie trailers with multiple genre labels such as action-comedy, or mystery-horror-thriller. Because of this, we have a multi-class problem, and have opted the one-vs-all approach. This means creating a classifier for one class, i.e. the drama genre, and simply identifying trailers as either drama (positive) or not drama (negative). For each genre, we passed our CSV spreadsheet to Matlab's fit binary classification decision tree function (*fitctree*) to build a decision tree. Only 80% of the trailers randomly selected from the full set of trailers were used for building the tree. This subset is known as a training set. The tree was then used to predict on the remaining 20% of trailers using Matlab's classification predict function (predict), and its success and failure rates were recorded. This process was repeated 40 times, each time with a new set of random trainer sets, with the success rates averaged, for the purpose of cross-validation.

We experimented with the support vector machine (SVM) model as well. The classifier was created using MATLAB's SVM fit function (fitcsvm), used for classification, the results were cross-validated (crossval), and their accuracies were noted (kfoldLoss).

IV. RESULTS

The rates of success of classification by our algorithm for top ten most popular genres are shown in Fig. 4. Success rates were calculated using both binary decision tree, and support vector machine sub-algorithms. For instance, the Drama genre had 58% and 66% successful classification for binary decision tree and SVM respectively. It had the lowest success rate of top-ten-genres, but the rest genres registered consistently over 70% success rate. SVM had roughly 8% higher success rate for a given genre than binary decision tree.

In order to gain some insight to what features are important

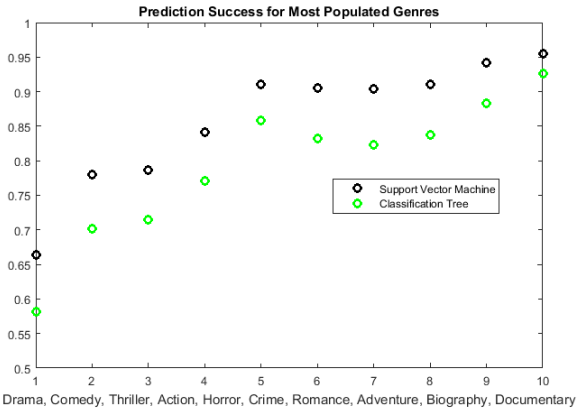

Fig. 4: Classifier success rate for ten most popular genres.

for the classification, we performed singular value decomposition on the set of movie trailers and their features. As can be seen in the plot of covariances of principal modes (fig. 6), one can see that there is no set of predominant modes that affect the decision, but its rather a complicated combination of all modes. The first four principal modes contain approximately 25% of energy in all modes. These four modes consist of features of differing degree. Figure **??** display the weights of each feature in four prominent modes. We could not identify any apparent patterns for any of these modes. We experimented with the concept of dimension reduction by

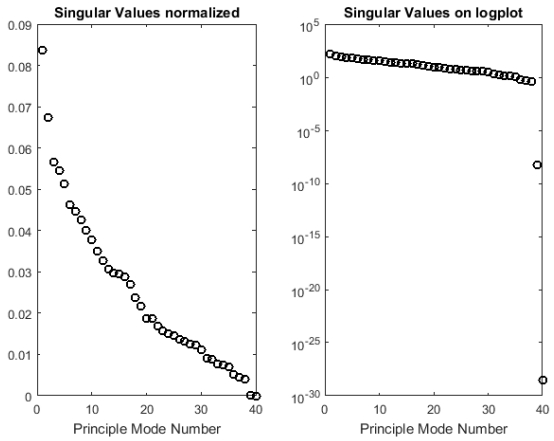

Fig. 5: Covariances of modes

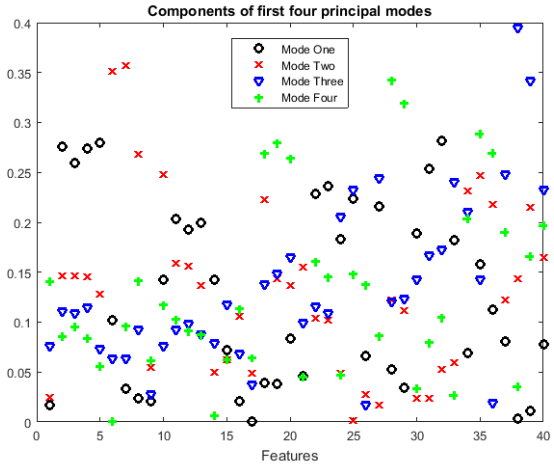

Fig. 6: Compositions of features for each modes

trying the classification using one, four and all of the modes. The rate of success are plotted in fig. 7. Surprisingly, the difference of success rate between one-mode classification and all-mode classifications was less than 10%.

V. LESSONS LEARNT

A. Standardization of Features

One crucial mistake that was overlooked during the initial phase of our development was negligence to standardize features. The original features had different units, and different raw values, often differing by order of magnitudes. For example, features such as 'total run-time' or 'number of shots' had values in the range of hundreds, while features such as 'min. volume' or 'min. shot length' had values in the magnitude of one decimal points. Without standardization, variations of values present in features with large raw values seem much greater to Matlab's trainer than those present in features with small raw values. In addition, the mean value of many of the features were not zero. This resulted in erroneously result where those features with large values showed up as modes largely driving the decision tree. Once

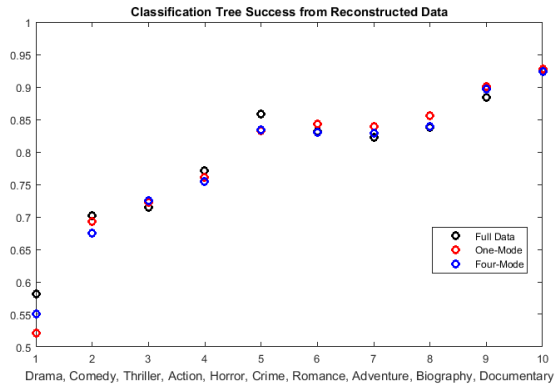

Fig. 7: Classifications made with one, two and four mode approximation compared to the original classification made with a complete set of modes

all the features were standardized, however, we achieved the results presented in the result section above. The principal modes and the composition of these modes became more complicated, but the algorithm treated features of different nature more fairly. The classification success rate was not affected much.

B. Unsuccessful Feature Extraction

One failed attempt to acquire features from the audio portion of movie trailers was to perform a Principal Component Analysis (PCA) using singular value decomposition. The goal of this process was to identify a series of principal modes and their components in each of the movie trailers. We hypothesized that the values of principal components could be used as features. We clipped 10-second portion of audio from each trailer to reduce the size of the data. Nonetheless, the matrix at which the singular value decomposition was to be performed had a size of 958-by-227150. This was computationally very expensive, with a runtime on the order of 10+ hours on modern personal computer. Furthermore, there was no guarantee that 10-second clipping would capture a signature sound of each trailer. (In a preliminary attempt, a 10-second clip was clipped at the timestamp of $t = \frac{total\ time}{2}$). Given these reasons, the attempt was deemed implausible in the sense of cost-benefit, and was abandoned.

VI. FUTURE WORK

An interesting topic of future work is comparing our results with the performance of human subjects. Since the definition of a genre is rather loose, it is often the case that there are many right answers for the genre of a given trailer. Thus, it is likely that human subjects guesses could differ from the genre labels in our metadata. We hypothesize that it may even be possible for machine learning to outperform humans at this task.

VII. CONCLUSION

This work demonstrated the construction of a movie genre classifier using features acquired from video and audio portions of movie trailers. A series of video analyses using a

computer vision library generated numerous video features, such as detail scores, dark scenes, color and intensity profiles, etc. Temporal and frequency analysis of audio identified sharp increases and decreases of volume and frequency spectra. MATLAB classifiers trained using these features classified movie genres with success rate ranging from 60% to 95%. Of two classifiers, the binary decision tree and support vector machine, the latter had approx. an 8% higher likelihood of successful classification overall.

It is worth noting that our machine learning algorithms pipeline has a minimum amount of required human interaction. The only step within that necessitates human input is providing a genre classification for the initial set of movie trailers used for training. The rest of the pipeline is almost entirely automated. While in this project we tested our algorithm using movie trailers, it can be adapted to any environment that requires computer vision-audio based machine learning. This can include surveillance, video tracking, and other applications.

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)
```

```
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)


import csv
import unicodedata
```

```
import re
import sys

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

    return movie_title

# Movie metadata csv file
csv_filename = 'movies.csv'
# File containing movie attribute metrics from video analy
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
analysis_dict = {}

with open(video_analysis_filename) as analysi
    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(':')[-
    avg_intensity = float(movie_lines[3].split(':')
    avg_color_r = float(movie_lines[4].split(':')[
    avg_color_g = float(movie_lines[4].split(':')[
    avg_color_b = float(movie_lines[4].split(':')[
    mean_shot_length = float(movie_lines[5].split(
    std_dev_shot_length = float(movie_lines[6].spl
    max_shot_length = float(movie_lines[7].split('
    min_shot_length = float(movie_lines[8].split('
    num_shots = int(movie_lines[9].split(':')[-1])
    stddev_color_with_letterbox_r = float(movie_li
    stddev_color_with_letterbox_g = float(movie_li
    stddev_color_with_letterbox_b = float(movie_li
    detail_score_mean = float(movie_lines[11].spli
    detail_score_std_dev = float(movie_lines[12].s
    detail_score_max = float(movie_lines[13].split
    detail_score_min = float(movie_lines[14].split
    dark_scene_mean_length = float(movie_lines[15]
    dark_scene_length_std_dev = float(movie_lines[
```

```
    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 lines
    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'] = sudd
    movie_dict['sudden_fall_count_per_cut'] = sudd

    # Add this dictionary to the dictionary of all
    if title in analysis_dict:
        for key in movie_dict.keys():
            # Add everything to the analysis_dict
            # except for the title, which has alrea
            if key != 'title':
                analysis_dict[title]['mean_volume'
                analysis_dict[title]['std_dev_volum
                analysis_dict[title]['min_volume']
                analysis_dict[title]['max_volume']
                analysis_dict[title]['sudden_rise_
                analysis_dict[title]['sudden_fall_
        analysis_dict[title] = movie_dict

    # Remove this movie (and the trailing blank li
    lines = lines[8:]

# Now, we've created a dictionary of all of th
# 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 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 "nicel
        # that match the version we pulled from the oth
        not_as_nice_title = format_title_for_file_s

        # Now, we'll check to see if we match anyt
        # 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
            analysis_dict[not_as_nice_title]['orig
            analysis_dict[not_as_nice_title]['mov
            analysis_dict[not_as_nice_title]['genr
            analysis_dict[not_as_nice_title]['rele
            analysis_dict[not_as_nice_title]['imdb

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

# 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_
        'detail_score_min' , 'dark_scene_mean_length', 'dark_scene_length_stddev', 'dark_scen
        'dark_scene_length_min', 'dark_scene_count', 'dark_scene_percentage',
        'genre', 'release_date', 'imdb_url', 'mean_volume', 'std_dev_volume',
        'min_volume', 'max_volume', 'sudden_rise_count', 'sudden_fall_count']
    writer = csv.DictWriter(csv_file, fieldnames=fieldnames)

    writer.writeheader()

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


import os
import sys

video_dir = 'C:\\Users\\Noel_K\\git_repository\\video-analysis-amath-582\\video_processing\\v
mp3_dir = 'C:\\Users\\Noel_K\\git_repository\\video-analysis-amath-582\\video_processing\\mp3

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_d

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()


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\\mp3
clipped_mp3_dir = 'C:\\Users\\Noel_K\\git_repository\\video-analysis-amath-582\\video_processi
```

```
clip_size_ms = 10000

def clip_and_save_audio(full_pathname, filename):

    sound = AudioSegment.from_mp3(full_pathname)

    # done and slicing 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('.')[:-
    print "Saving %s as %s" % (full_pathname, os.p

    clip_samples.export(os.path.join(clipped_mp3_d

    print full_pathname

def main():
    # Go through all directories
    for dirname, dirnames, filenames in os.walk(mp
        # 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(d

if __name__ == '__main__':
    main()


# -*- 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
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 w



#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_co
    writer = csv.DictWriter(csv_file, fieldnames=fieldnames)

    writer.writeheader()
```

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

```
%% Import data from text file.
% Script for importing data from the following tex
%
%    C:\Users\johnf_000\Dropbox\MATLAB\Assignments
%
% To extend the code to different selected data or
% 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\Assi
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

%% 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, '

%% 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_saturation avg_color_variance ...
    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_stddev ...
    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_fall_count ...
    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([0 11 0 1])
title('Decision Tree Prediction Success for Most Popular 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', '
title('Singular Values normalized'), xlabel('Princ
subplot(1,2,2), semilogy(lambda, 'ko', 'LineWidth'
title('Singular Values'), xlabel('Princ
%% Building Reconstructions
ff = u'*master_data; %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+', 'LineWidt
title('Components of first four principle modes');


%% Testing reconstruction
clearvars accuracyOne
accuracyOne = [];
data = ff1';
num_trials = 40; % number of cross validation tria
accuracyOne(1) = check_genre_predictions(data, Dra
accuracyOne(2) = check_genre_predictions(data, Com
accuracyOne(3) = check_genre_predictions(data, Thr
accuracyOne(4) = check_genre_predictions(data, Act
accuracyOne(5) = check_genre_predictions(data, Hor
accuracyOne(6) = check_genre_predictions(data, Cri
accuracyOne(7) = check_genre_predictions(data, Rom
accuracyOne(8) = check_genre_predictions(data, Adv
accuracyOne(9) = check_genre_predictions(data, Bio
accuracyOne(10) = check_genre_predictions(data, Do

%% Another Reconstruction test
clearvars accuracyTwo
accuracyTwo = [];
data = ff4';
num_trials = 40; % number of cross validation tria
accuracyTwo(1) = check_genre_predictions(data, Dra
accuracyTwo(2) = check_genre_predictions(data, Com
accuracyTwo(3) = check_genre_predictions(data, Thr
accuracyTwo(4) = check_genre_predictions(data, Act
```

```
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 Recursive 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 label Drama, Comedy, Thriller, Action, Horror, C
% 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
CVSVMModel = crossval(SVMModel);
accuracySVM(6) = kfoldLoss(CVSVMModel);

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


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

SVMModel = fitcsvm(master_data, Biography, 'Standa
CVSVMModel = crossval(SVMModel);
accuracySVM(9) = kfoldLoss(CVSVMModel);
SVMModel = fitcsvm(master_data, Documentary, 'Stan
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
title('SVM Prediction Success for Most Populated G
xlabel('Drama, Comedy, Thriller, Action, Horror, C

%% SVM vs Tree
figure(3)
clearvars success
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, Docume
legend( 'Support Vector Machine','Classification Tree', 'Location', 'best')


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);
```

VIII. ACKNOWLEDGMENTS

We would like to thank Dr. Nathan Kutz for his advice and feedback on our techniques used in this project. We'd also like to thank Kam Chancellor, because Nathan would want us to.

REFERENCES

[1] Rasheed Z., Sheikh Y., Shah M., *On the Use of Computable Features for Film Classification*, IEEE Transactions on Circuits and Systems for Video Technology, Vol.15 No.1, Jan. 2005.
[2] *OpenCV — OpenCV*. Itseez. Web. 09 Mar. 2016. http://opencv.org/.
[3] Canny, J., *A Computational Approach To Edge Detection*, IEEE Trans. Pattern Analysis and Machine Intelligence, 8(6):679698, 1986.