Stanford University

2016 Autumn CS229

Machine Learning

Project Final Report

Due on Dec 16th, 2016

Author:
Bojiong Ni (bojiong)
David Wugofski (dwugo)
Zhiming Shi (zhimings)

Instructor:
Andrew Ng
John Duchi

By turning in this assignment, we agree by the Stanford Honor Code
and declare that all of this is our own work.

Dec. 15th, 2016

VIDEO GAME GENRE CLASSIFICATION USING VIDEO GAME MUSIC

BOJIONG NI, DAVID WUGOFSKI, ZHIMING SHI

1. Problem Definition

Our project will attempt to use the soundtracks of a video game to classify the game's genre. While music genre classification is a well-studied problem, classifying a video game's genre off of its soundtrack is not an area that has received attention. While there has been heuristic analysis of what makes a soundtrack work for a given video game genre, no algorithms currently exist which attempt to take a soundtrack and estimate the video game's genre from that data. Video games represent very deep, high complexity sets of data (with sounds, images, story events, etc) and few algorithms exist which attempt to analyze them. Songs, on the other hand, are well studied, and many models exist for analysis. We believe that by analyzing a video game's soundtrack along, we should be able to gain substantial insight into the video game's genre.

2. Data

We've built our own labelled dataset from MIRSoft's database[1] of video game music files, which includes the complete soundtracks. We've crawled and downloaded the data of all 53,630 video games, out of which only 7900 games have music provided on the site. Furthermore, only 1349 games had music files formatted in ways we could process. Each game has one or several pieces of music, with a meta data file containing information of its release and content, including genre. We've analyzed the genres of all games, and decide to manually re-define genres based on the originally provided ones. The genres we define are unambiguous, well-understood, and each contains a workable size of games:

Genre	# Games	Example
Logic, Puzzle	222	Mah Jong Solitaire
RPG, Fantasy	225	Final Fantasy series
Platformer (Jump n Run)	343	Super Mario
Fight	161	Street Fighter
Sports	59	Fifa Soccer
Racing	70	Fomula 1
Strategy	62	Castles
First Person Shooting	183	Dark Forces

3. Feature Extraction

We used two forms of features for our learning algorithms. The first feature vector layout contains features for individual songs. For this vector we chose features commonly used in musical genre classification, laid out in [2] and [3]. Our second feature vector combines all the song feature vectors from a soundtrack into one video game feature vector.

3.1. Song Feature Vector.

The commonly-used features for musical genre classification are studied in [2] and [3]. We've provided an explicit, ordered outline of the features we extract in table 1 (in Appendix A.), contrasting that with Silla's feature vector[2] in table 2. We have a total of 46 features in our vector. The feature vector is divided into three categories: *timbral* features, *rhythmic* features, and *pitch* features. The timbral features outline the frequency-spectral behavior of the music as it varies with time. The rhythmic features attempt to quantify the general rate at which notes are played in the music. The pitch features attempt to quantify which notes are favored over others in a musical composition.

For most of these features, direct feature calculations are made over a small window of the song. The average and standard deviation of these features are then calculated over the course of the song. These averages and standard deviations are the actual features included in the song vector.

3.1.1. *Timbral Features.*

The timbral feature are meant to capture how the musical composition is distributed spectrally. While pitch features are concerned with which notes are being played, timbral features are related to which instruments are being used to play the notes of the piece. While the pitch features would be relatively insensitive to whether a sequence of notes was played on a piano or guitar, the timbral features would be greatly affected. The song features relevant to timbre are

(1) Spectral Centroid: The center of gravity for the frequency spectrum. The spectral centroid can be thought of as the expected value of the spectral distribution of a frame. Larger values of the centroid reflect a bias towards higher frequencies, whereas smaller values reflect a bias towards lower frequencies.

(2) Spectral Bandwidth: The spectral bandwidth at frame t is the mean-squared difference between the frequency spectrum and its centroid. If we consider the spectral centroid to be the expected value of the spectral distribution of a frame, the spectral bandwidth can

Date: December 17, 2016.

be considered a version of the variance of the spectral distribution of a frame. Large values of the spectral bandwidth equate to a spectrally broad frame, while small values equate to a spectrally narrow frame.

(3) Spectral Rolloff: The spectral rolloff is the frequency bin below which 85% of the spectrum magnitude distribution is concentrated.

(4) Zero Crossing Rate: The zero crossing rate is the number of times the time-domain signal crosses zero in a window frame.

(5) Mel-Frequency Cepstral Coefficients: The Mel-Frequency Cepstral Coefficients (MFCCs) are a perceptually-motivated modified version of the STFT. MFCCs capture spectral perception better than the STFT two reasons. First, the MFC is a ceptstum: a frequency spectrum of a frequency spectrum. It measures the repetition and periodicity of frequencies. Second, the mel scale mapping is meant to approximate how humans recognize the spacing of notes, allowing the MFC to be a better cepstrum than one taken directly from the STFT.

According to results obtained in [3], only the first five MFCCs are relevant for musical genre classification. For this reason, we have chosen only to add the first five MFCCs to our feature vector.

(6) Spectral Contrast Coefficients: The idea behind using Spectral Contrast Coefficients for music genre classification is outlined in [4]. While MFCCs measure the periodicity of the spectrum of a frame, the SCCs measure the roughness of the spectrum when broken into different octaves. Another way to view these features is as a measure of which bands contain the most spectral activity for a given frame.

(7) Spectral Flux: The spectral flux of a frame is the sum of the squared distances between the normalized magnitudes of successive frequency bins. It is a measure of how rapidly the spectrum changes in frequency.

(8) Low-Energy Feature: The low energy feature is defined as the fraction of frames whose RMS value is below the RMS value of the song as a whole. The low-energy feature measures how concentrated the energy of the song is with respect to time.

3.1.2. *Rhythm Features.* The rhythm features extract information on the timing, beat, and tempo of the song. It measures the rate at which notes are played back and the speed of the music.

The first step in calculating the rhythm features is constructing the beat histogram of the song. The beat histogram is a measure of how dominant certain beats are in the song.

For our beat histogram, we take the time average of the tempogram of our song calculated according to the algorithm laid out in [5]. Using sudden changes in spectral content between frames, the tempogram algorithm first creates a "novelty curve" which peaks in intensity on the onset of notes. Using a windowed autocorrelation algorithm on the novelty curve, the algorithm creates a 2-D curve of which tempos most accurately reflect the underlying periodicity of the piece of music. This curve is the tempogram of the signal.

We then average the tempogram across frames to obtain our beat histogram. With this histogram we locate the two largest peaks. We find their amplitudes and their periods. As well, we calculate the sum of the beat histogram to measure the rhythmic intensity of the song.

3.1.3. *Pitch Features.* The pitch features measure which notes are preferred in a musical composition. Much like the key figure of merit for the rhythmic features is the beat histogram calculated from the tempogram, the key figure of merit for the pitch features is the pitch histogram calculated from the chromagram. However unlike rhythm, because notes can be played in different octaves, there is a distinction made between an octave-less, folded pitch histogram and an unfolded pitch histogram which spans multiple octaves.

Our formation of the pitch histograms begins with the implementation of a constant-q transform applied to the signal, as described in [6]. The constant-q transform is a variation on spectral frequency representation with frequency bins spaced out in octaves. This is ideal for pitch calculations, as notes are likewise spaced in octaves. Thus you can think of each frequency bin as representing one note. We then map each bin in the CQT to the same octave, creating a chromagram: a representation of the strength of presence of each "letter" note, regardless of octave.

After we have performed this map, we then perform an energy normalization method prescribed in [7]. This method smooths out the chromagram perceptually. It attempts to better catch gradual changes in pitch and remove noise while better matching human perception.

By averaging the constant-q transform across all frames, we get a measure of which notes from which octaves were favored: the unfolded pitch histogram. Likewise, by averaging the chromagram across all frames, we get a measure of which "letter" notes were favored, regardless of their octave: the folded histogram. We take the period of the dominant pitch from the unfolded histogram, the amplitude and period of the most dominant pitch from the folded histogram, the difference in pitch number (i.e. frequency) between the most and second-most prominant pitches of the folded histogram, and the sum of the pitch histogram. The sum of the pitch histogram measures the overall intensity of the song.

3.1.4. *Implementation.* We implemented song feature extraction using the LibROSA python library [8]. For spectral features, all except the spectral flux and low-energy feature are implemented in the LibROSA library. For rhythm and pitch features, the fundamental figures of merit (tempogram, constant-q transform, and chromagram) is implemented. In our implementation, we used Hann window frames of 512 samples and Fourier transforms of 2048 samples. For the constant-q transform we

used a fundamental frequency of 16.35 Hz (C0) and 8 octaves.

3.2. **Aggregate Feature Vector.** Once all the features for the individual songs had been extracted, we attempted to create an aggregate vector composed of the features from the individual songs. We decided to start simply by taking the mean and standard deviation of each feature in the song feature vectors across the entire soundtrack. The resulting vector contained 92 features: two for each feature in the song vectors. For the purposes of training and testing, we did not consider video games with only one song in their soundtrack, as all the standard deviation features in the aggregate vector would necessarily be zero in such cases.

After creating these 92 "simple" aggregate features, we wanted to try devising other features to possibly increase learning performance. One option we considered and had time to implement was a form of k-means clustering features. For these features, we ran k-means clustering on all our song feature vectors, creating 6 clusters for the songs. For each game, we calculated how many of its songs mapped to each cluster. We normalized these numbers by the total number of songs in the soundtrack, and added the 6 numeric results to the aggregate feature vector. In total, our aggregate feature vectors contained 98 features.

4. Game Genre Classification

In the original data set, some game genres have a small number of samples. We hand picked 5 game genre with at least 96 games for our projects. We threshold by requiring each game to have at least 3 songs in the soundtrack, hence our approach in feature aggregation would be more meaningfully applicable.

4.1. **Classification Algorithm.** We have two high level approaches for this problem. The first approach is to train the game genre on each of the songs from the game. At prediction time, each song from the game will receive a prediction of genre. Then we run a majority vote scheme where the genre with highest votes from the songs will be selected as the genre of the game.

In the second approach, we aggregated the music features from each song in the game to create one feature vector per game. Then train and test on the game vector.

The aggregated game feature contains the mean and standard deviation of all music features from the game. In addition, we ran a k-means clustering for all songs from all games. Then for each game, we create a vector of size k, where each element is a histogram indicating how many songs from this game belongs to each of the k clusters. We then normalize this vector and concatenate it together the mean vector and standard deviation vector.

In both approaches, we trained trained four different models:

- **Gaussian Naive Bayes** Since our feature vector is real valued. One strategy is to model each feature with Gaussian distribution. Using the Naive Bayes classifier discussed in class, we can extend the classifier to multi class instead of just binary.
- **SVM one vs one** SVM one vs one is to fit one SVM linear classifier per class (genre) pair. At the prediction time, the class receiving the highest vote is selected as prediction label.
- **SVM one vs rest** SVM one vs rest is to fit one classifier per class (genre). At prediction time, the class with highest confidence score is marked as prediction label. For both SVM algorithms, we've experimented on different kernels, and we concluded that the linear kernel achieves the best classification results.
- **Deep Neural Network** The DNN we used has three layers, with 10, 20, 20 hidden units in each layer. We've experimented on different configurations of the DNN, including the number of layers, number of units, and the step time, this achieved the best result.

4.2. **Results and Evaluation.** To We hold off 20% of our selected dataset as a test set for evaluation. We've looked at 2 different evaluation metric: the test accuracy and the F1 score which combines both precision and recall. We realized that the F1 score are all pretty close numerically to our test accuracy, therefore, for presentation in this report, we only tabulate our test accuracy:

Learning	2 classes		3 classes		5 classes	
method	Vote	Agg	Vote	Agg	Vote	Agg
Softm	60.5%	60.5%	61.4%	61.4%	25.3%	42.1%
NB	55.3%	50.0%	49.1%	50.9%	25.3%	29.5%
SVM 1v1	50.0%	60.5%	40.4%	47.4%	22.1%	26.3%
SVM 1vA	50.0%	60.4%	33.3%	36.8%	33.7%	22.1%
DNN	–	–	52.6%	45.6%	20.0%	34.7%

Above, we've tabulated the results of classification using both approaches of our feature extraction: individual songs with voting, and using aggregated sound track feature vectors for each game. We've carried out classification using 2 classes (Puzzle and RPG), 3 classes (Puzzle, RPG, Platformer), and 5 classes (Puzzle, RPG, Platformer, First-person shooting, and strategy).

First, note that in most cases, classification using aggregate features out performs that using individual songs with voting. This confirms our hypothesis that the soundtrack as a whole collectively captures more characteristics of the game. Although the overall accuracy, even by our best algorithm attempt, which is softmax on 5-class dataset, is not that high, we'd still consider this attempt somewhat successful, as we achieved good improvement over the bayes classifier, which minimizes the error of misclassification. In this case, we still manage to out-perform the baseline. It's worth noting that in our DNN, we occasionally observe over-fitting, with training accuracy as

high as 82%. We've performed principle component analysis with singular value decomposition, and realized that our aggregate features has around 16 principle components. We've also attempted dimensionality reduction by projecting our feature vectors into this reduced space, the results is not significantly better.
With the above said, our final accuracy is not as satisfactory as we set out to believe. We've manually evaluated a few songs, from different classes, and identified some problems that our aggregate feature vector fails to capture. For example, there is a distinction between game entrance music, and background music. The game entrance music sounds pretty similar for a puzzle game or a shooting game. This suggests some inadequecy still in our feature vectors, which we shall discuss below.

5. Discussion And Further Work

Our results show promise for using soundtracks to estimate video game genre, though there is still room for improvement. Our best results were achieved with Softmax Regression using our aggregate feature vector. We believe that, with further improvements to our feature aggregation, we can improve the performance of our algorithms. We have also considered improving the individual song classification algorithm through the use of a different cost function. As well, we considered the practicality of reducing the dimensionality of our feature vectors to improve performance.

Given more time, we would like to increase our dataset size and consider other methods of feature aggregation. We eliminated many viable games simply because their file formats were not widely supported. Give more time, we would like to track down methods of converting or directly processing those songs to dramatically increase the number of games we can use as samples.

Additionally, we have considered other methods of feature aggregation we have not had time to implement. From listening to songs in our database, we find that the general feel of a song is much more indicative of the video game setting rather than the genre. We believe that, when a video game soundtrack is being composed, song features are chose to represent certain "moments" in the video game - menu screens, cutscenes, action sequences, firefights, etc. - rather than the video game as a whole. Thus we expect that the optimal aggregate feature vector contains features relating to the presence of these moments in the game: a puzzle game would have quite a few "menu" moment songs, while a first person shooter may have a lot of "cutscene" and "firefight" moments.

While we attempted to use our k-means features to express this, we did not find them to significantly improve our results. Thus we would like to investigate other methods of better expressing these moment distributions. One option we considered is to use the results from individual song classification as features for our aggregate feature vector. If the key characteristic for song to game genre estimation is in fact the moment the song is meant to represent, the results of our individual song classification algorithm would be closely correlated with the moments each song represents. At the very least, this method would ensure the aggregate feature vectors performed no worse than the individual song vectors.

While we are focused on improving aggregate feature vectors, since those vectors produce the best results, we have also considered using a different objective function for our individual song classification. Our individual song classification methods attempt to classify all songs according to genre. However we only need for a plurality of songs within a game to be classified as belonging to the correct genre. The objective functions used in this project do not account for the fact it is okay to misclassify songs, provided the genre with the most songs in a video game is the genre of the game. By devising an objective function which would take this voting method into account, we expect we could improve the performance of individual song classification.

Lastly, we also wish to look into running dimensional reduction on our current feature vectors. From analyzing the k-means clusters, we found that the centroids only varied significantly in about 5-10 dimensions, suggesting very little variation in our data along those dimensions. Additionally, running PCA on our song feature vectors, we found that only about 15 dimensions were necessary to capture the variance of our data. These results indicate we have a lot of redundancy in our feature vector we could eliminate by running PCA and producing a new set of song vectors. As well, for the purposes of determining aggregate features, it may be relevant to manually determine how much each feature improves the performance of our learning algorithm, and remove features which do not significantly increase performance.

References

[1] (2016). Game music database, *World of game Music*, [Online]. Available: `http://www.mirsoft.info/gmb/index.php`.

[2] C. Silla, A. Koerich, and C. Kaestner, "A machine learning approach to automatic music genre classification", *Journal of the Brazilian Computer Society*, vol. 14, no. 3, 2008. DOI: `http://dx.doi.org/10.1007/BF03192561`.

[3] G. Tzanetakis and P. Cook, "Musical genre classification of audio signals", *IEEE Transactions on Speech and Audio Processing*, vol. 10, no. 5, 2002.

[4] D.-N. Jiang, L. Lu, H.-J. Zhang, J.-H. Tao, and L.-H. Cai, "Music type classification by spectral contrast feature", in *Proceedings. IEEE International Conference on Multimedia and Expo*, vol. 1, 2002, 113–116 vol.1. DOI: `10.1109/ICME.2002.1035731`.

[5] P. Grosche, M. Muller, and F. Kurth, "Cyclic tempogram - a mid-level tempo representation for musicsignals", in *ICASSP*, IEEE, 2010, pp. 5522–5525, ISBN: 978-1-4244-4296-6.

[6] C. Schorkhuber and A. Klapuri, "Constant-q transform toolbox for music processing", in *SMC Conference*, IEEE, 2010.

[7] M. Muller, F. Kurth, and M. Clausen, "Audio matching via chroma-based statistical features", in *Proceedings. International Conference on Music Information Retrieval*, ISMIR, 2005.

[8] (2016). LibROSA documentation, [Online]. Available: `http://librosa.github.io/librosa/index.html`.

Appendix A. Features

E-mail address: `bojiong@stanford.edu, dwugo@stanford.edu, zhimings@stanford.edu`

TABLE 1. Our feature vector

Timbral features
Spectral centroid average
Spectral centroid std. dev.
Spectral bandwidth average
Spectral bandwidth std. dev.
Spectral rolloff average
Spectral rolloff std. dev.
Zero-crossing rate average
Zero-crossing rate std. dev.
MFCC #1 average
MFCC #2 average
MFCC #3 average
MFCC #4 average
MFCC #5 average
MFCC #1 std. dev.
MFCC #2 std. dev.
MFCC #3 std. dev.
MFCC #4 std. dev.
MFCC #5 std. dev.
SCC 1^{st} octave average
SCC 2^{nd} octave average
SCC 3^{rd} octave average
SCC 4^{th} octave average
SCC 5^{th} octave average
SCC 6^{th} octave average
SCC 7^{th} octave average
SCC 1^{st} octave std. dev
SCC 2^{nd} octave std. dev
SCC 3^{rd} octave std. dev
SCC 4^{th} octave std. dev
SCC 5^{th} octave std. dev
SCC 6^{th} octave std. dev
SCC 7^{th} octave std. dev
Spectral flux average
Spectral flux std. dev
Low-energy feature

Rhythmic features
Relative amplitude of first beat histogram peak
Relative amplitude of second beat histogram peak
Period of first beat histogram peak
Period of second beat histogram peak
Ratio of first peak amplitude to second peak amplitude
Sum of beat histogram

Pitch features
Relative amplitude of folded pitch histogram peak
Period of first unfolded pitch histogram peak
Period of first folded pitch histogram peak
Pitch interval between first and second peaks of folded pitch histogram
Sum of pitch histogram

TABLE 2. Silla's feature vector from [2]

Timbral features
Spectral centroid average
Spectral rolloff average
Spectral flux average
Zero-crossing rate average
Spectral centroid std. dev.
Spectral rolloff std. dev.
Spectral flux std. dev
Zero-crossing rate std. dev.
Low-energy feature
MFCC #1 average
MFCC #2 average
MFCC #3 average
MFCC #4 average
MFCC #5 average
MFCC #1 std. dev.
MFCC #2 std. dev.
MFCC #3 std. dev.
MFCC #4 std. dev.
MFCC #5 std. dev.

Rhythmic features
Relative amplitude of first beat histogram peak
Relative amplitude of second beat histogram peak
Period of first beat histogram peak
Period of second beat histogram peak
Ratio of first peak amplitude to second peak amplitude
Sum of beat histogram

Pitch features
Sum of pitch histogram
Period of first unfolded pitch histogram peak
Relative amplitude of folded pitch histogram peak
Period of first folded pitch histogram peak
Pitch interval between first and second peaks of folded pitch histogram