Creating Video Highlights from Chat Logs

Christopher Zhang Cui, Cole Hopfenberg, Mike Kelly, Ami Zou
{ccui9, colehopf, mckelly, yixuanz}@live.unc.edu

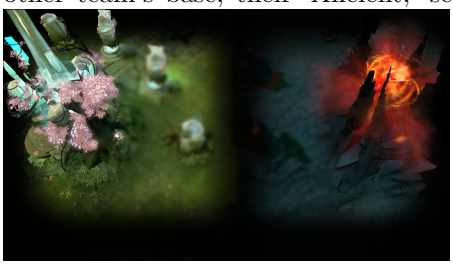
Abstract

Twitch.tv is a streaming site where people can stream themselves playing games with viewers often watching and interacting with the streamer through a chat window. By analyzing these chat logs and training two Linear Regression models to recognize emotes and parse times of high chat activity respectively, we present a method of classifying parts of a video a viewer might find exciting, funny, sad or otherwise worth watching. Our data-set is based off chat logs obtained from streams of the popular Multiplayer Online Battle Arena(MOBA) game, Dota 2.

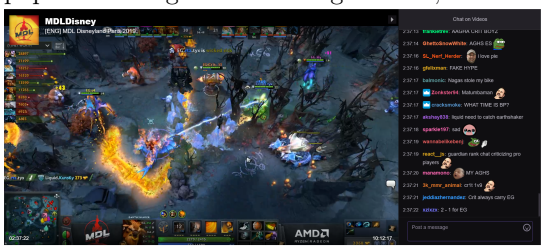
1 Introduction

For our project, our group chose to create a program that uses linear regression to identify significant clips from games of Dota 2 that are streamed online.

In Dota 2, ten players, each controlling a unique hero, are grouped into two teams. Each team's main objective is to destroy the other team's base, their 'Ancient,' seen below.



Typically, the most exciting moments are when both groups of heroes gather and attempt to kill the heroes on the other team. These 'teamfights' are typically when a player's skill is on greatest display and produce the most exciting highlights. And it is on these highlights that we focused our efforts on identifying. Our program analyzed recent chat-logs from Dota 2 Tournament games that were streamed on the popular video game streaming website, Twitch.tv.



We chose Dota 2 for our test cases as we were personally familiar with the game. Although we only used games of Dota 2 on twitch, our hope was to create a program that could be generalized to any game that is streamed with a live chat.

2 Related Work

PogChampNet is a similar application that predicts the game highlights given a video and its chat-log. It uses InceptionResNetV2, a deep convolutional neural network, to train the data. For assigning the labels, the authors give a score from 0-9 to each 1-second clip for deciding how excited/highlight-worthy this clip is. They use Karas and Python for training the data, and train it for about 12 hours on 20 epochs. The testing error is fairly low.

The UNC research project is also a game highlight prediction network. It uses CNN-RNN

model architecture: V- CNN-LSTM for the vision model; L-Char-LSTM for the language model; and modellv-LSTM for the joint multimodal. They train on 218+103 = 321 videos and have an accuracy varying from 70.0% to 74.7% (the L-Char-LSTM performs better than L-Word-LSTM).

3 Methods

3.1 Emote Model

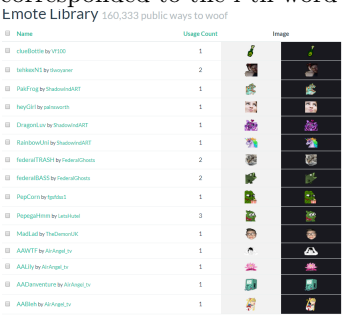
On twitch, emotes are commands that when entered into the chat log, appear as a message. In the image above, the picture of a man putting a hand to his face is the command 'FailFish' while the image of another man's laughing face is 'EleGiggle'. Typically, after or during a highlight, the chat will see a great increase in messages that include emotes, so we believed the number of emotes that appeared during a given period would be a good parameter to include in our clip clip creator.

For the emote model, we began by loading a large sample set of chat messages into our python notebook. The program ***Twitch Chat Downloader*** allowed us to download entire chat logs as text files. We then formatted the text file into a new array that served as our feature vector. Each unique word was added as a sample, and for each word(W) we counted the number of times that the word appeared in the chat(A), the length of the word(B), the number of capital letters(C), the average number of repetitions(D), the number of lowercase letters that appeared before the first capital letter, set to zero if the word is entirely uppercase or has no uppercase letters(E), and the number of times that the word appears in a message by itself(F). Our feature vectors were of the form:

[W, A, B, C, D, E, F]
['Yooo', 1, 4, 1, 1.0, 0, 1]
...

Indexes 0-5 of our feature vector array. Index 0 is formatted out so we are left with a X by 6 feature vector array

Next to train the weights, we created a master list of emotes used by Twitch.tv, BTTV, and the channels that we believed to be most popular, and cross-referenced that with the words matrix to obtain a column vector y where the i-th element in y corresponded to the i-th word in our data matrix.



With over 160,000 emotes from the popular plug in Better-Twitch-TV(BTTV) alone and the countless channels with their own subscriber emotes, maintaining, let alone creating, a master-list of emotes is nigh impossible

We chose linear regression to train the emote model to simply classify a word as an emote command or not an emote command. For this step, we chose the logistic regression for 0, 1 classifications. In this model, we do not include a bias term as, by design, there will never be a feature vector with all parameters of zero. In the case where the parameters are closest to zero, such a 'word' is more likely an error than anything and the chance it is an emote should be as close to zero as possible. Thus, we do not include an initial bias term.

$$\sum_i y_i(\ x_i^T \beta) - \log\left\{1 + (\ x_i^T \beta)\right\} - \frac{\lambda}{2}\beta^2$$

Our penalized log-likelihood

Taking the derivative with respect to the weights gave an equation to use for gradient ascent.

$$\sum_i y_i\ x_i^T\ w_i - \frac{(\ x_i^T)(\ w_i)}{1 + (\ x_i^T \beta)} - \lambda\beta$$

Derivative in terms of w_i

To calculate our loss, we multiplied our 4542 by 6 data matrix by our 6 by 1 weight matrix to get a 4542 by 1 matrix of values between 0 and 1. For each of these values, if the value was less than .5 it was set to 0, and if the value was greater than .5, it was set to 1. Then, we compared each element with its corresponding value in our column vector y and incremented our loss by 1 for each value that did not match.

$$MAE = \frac{\sum_{i=1}^{n} |y_i - y_i^p|}{n}$$

Our loss function

3.2 Emote Model Results

We tested our weights trained through gradient ascent against a matrix of randomized weights and found the matrix of trained weights to be even less accurate than the randomized weight matrix.

Weights	Loss
Randomized	.7736
Trained	.9775

Our initial data set was a collection of 4542 unique words that appeared in a chat log over a five hour period

Faced with a model that increased our loss after training, we tried flipping the process and using gradient descent instead to fantastic results.

Weights	Loss
Randomized	.7736
Trained(G.Ascent)	.9775
Trained(G.Descent)	.02246

A drastic improvement from an accuracy of over 95 percent

Restarting with an initial weight array of all ones to achieve a more stable model, we tested different alphas to find the one that produced the best results.

alpha	Loss
400	.0416
300	.0354
100	.0284
0	.0244

To our surprise, an alpha of 0 minimized our loss. All alphas were tested with 500 iterations

Next, with our hyper-parameter alpha set to zero, we tested different numbers of iterations to find a point where there was no point in doing further iterations.

Iterations	Loss
600	.0244
400	.0253
300	.0257

Tested with an alpha of 0. Past 500 iterations, even 100 additional iterations did not result in a lower loss.

Finally, the last area to adjust was our step size. Again, with our optimal alpha and number of iterations, we tested a number of different step sizes.

Step-size	Loss
1e-6	.0320
1e-5	.0250
1e-4	.0245
1e-3	.0241

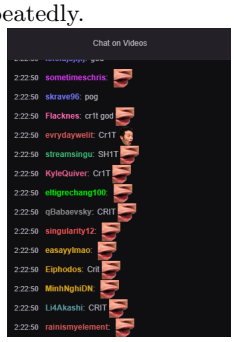
A step size of 1e-3 lent itself to the lowest loss. Anything lower resulted in the code failing to run

Testing on a secondary chat log of 13762 unique words resulted in a loss of only .0328(A total of 451 mis-classifications). With 500 iterations, a step size of 1e-3 and a hyper-parameter of 0, we next used these weights to train a model to detect clips.

3.3 Clip Model

For our clip model, the methods were almost identical to the emote model. The biggest difference was in how we formatted the data matrix. We parsed the data into intervals of 1 minute, with each interval corresponding to one row of our matrix. The columns of the matrix correspond to our different parameters. The parameters were the following: how many messages were in a given interval, the average length in characters of all messages within that interval, and the number of unique users commenting. We also used the weights derived from the emote model to count how many emotes occurred within a given minute. We chose these parameters because we

thought that overall they would be a good predictor of what is an important segment or not. For example, a large spike in the amount of messages sent is probably a good indicator of excitement, while a lull in the chat conversation will most likely mean that nothing too noteworthy is happening. Additionally, by counting the number of unique users, our algorithm is able to filter out any instances of a single user 'spamming,' or copying and pasting, messages repeatedly.



The chat immediately after a play by EG player Cr1t

Next, we fed the data we had gathered, in the form of an N-by-four list, into logistic zero-and-one regression, where N corresponds to how many intervals exist in the chat log. We first created a randomly-generated weights list with dimensions four-by-one in order to hold our weights as they are optimized. Next, our program ran two different optimization functions to train our weights. First we tried gradient ascent but found, as with the emote model, gradient descent worked better. We then ran prediction on both the weights trained from gradient ascent and the weights trained from gradient descent, and used the same loss function as in the emote model to compare it with the output.

3.4 Clip Model Results

Initially, our accuracy was 74%. This is because we deemed something worthy of being a highlight if all of its parameters were over 25% of the average for that column.

Our next way of training the data was more effective. First, we decreased the time interval from 1 minute to 15 seconds. This gave us more data points to better refine what was a clip and what wasn't. Additionally, instead of creating our training data by checking each row against the average, we manually decided what is a clip. With this combination of more data points and more accurate training data, our accuracy went up 10%.

We found that for the clip classifier model, the gradient descent optimization function resulted in a higher accuracy over the untrained weights and the weights trained using gradient ascent.

Method	Loss
G. Ascent	.661
Untrained	.677
G.Descent	.161

As with the emote model, we found that gradient descent succeeded in minimizing our loss

With 200 iterations, a step size of 1E-7 and again an alpha of zero, the weights served to allow our model to predict whether or not a given 15 seconds interval is a highlight of the stream or not. In a 47 minute game, we manually selected about 8 minutes of highlights and were able to detect about 6 minutes and 45 seconds of highlights. This model can, in theory, detect any type of clip on Twitch as long as the stream has chat enabled.

4 Conclusion

Our project set out to create a model that uses linear regression in order to identify significant clips from the popular game Dota 2, with the hopes that the model could be used for any game that is streamed on Twitch. This model can, in theory, detect any type of clip on Twitch as long as the stream has chat enabled. To give one example, it could be used in an automatic report system that detects inappropriate content or content that is against the Twitch Terms of Service. One could train the model using chatlogs from streams that were already reported and deemed inappropriate. This could lower the overhead cost of people having to manually check every stream that gets reported. The main difficulty encountered in designing our project was running into issues with linear regression and gradient ascent. In the future, this model could be further worked on by applying a clustering model to automatically categorize a clip as exciting, funny or sad.