Overview of our Project and Processes:

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. We hope this project can be used to automatically create highlight reels from any game it observes. Our program analyzes the chat from games that were streamed on the popular video game streaming website, Twitch. We chose Dota 2 for our test cases as the game is relatively complex, with a lot of different things going on in the game at any one particular time. Additionally, the game is very popular, so we knew there would be a lot of comments in the chat that we could use for our model. 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.

We began by creating a model to identify emotes in the chat, as these are, we believe, the best predictors of the tenor of the game at any particular moment. Twitch creates emotes from input strings, with the user typing in a specific string, and Twitch displaying the corresponding image to the chat. Our model was built to learn which strings correspond to such images. We believe that this approach is more effective than simply relying on a master list of emotes, as new emotes are often added to these sites, which would require a large amount of upkeep if we were to rely on manual input of emote strings. Additionally, there are channels on Twitch which have their own emotes, making manual upkeep of a list of emotes virtually impossible. We used a classification model using the following parameters: length of the string, capitalization, and frequency of appearance. We then used our built emote model to gather the data which would be analyzed in the next part of our project: the clip classifier.

Our clip classifier divides a given video into one-minute segments to be analyzed. The segment is then given a designation as to whether or not it is a relevant clip. This part of the

project was done by hand as to then be able to increase the scope of what our model could predict using given emotes.

Next, we combined the emote model with the clip creator to make a third model that groups emotes according to their classification from the section above. This was done using clustering. Finally, we used our newly-classified emotes to create significant clips. With this new information, we were able to not only find the timestamps of said clips, but also whether they were happy, sad, funny, etc.

Details of Implementation:

For the emote model, we began by loading a large sample set of chat messages into our python notebook. We then put the text into a double array, with each message being an index in the first array, and each word in the message being an index in the second array. Next, our program counted how many unique words were in the chat log, removed their usernames and timestamps, which were still stored in the first two elements in each subarray, and added each word to a new array. Each word was only added once, regardless of how many times it appeared in the chat.

Using our newly-created words array, the program then went back through the chat and created a new words matrix, with each word serving as a column. The parameters for each word were the number of times that each word appeared in the chat, the number of capital letters per word, the number of lowercase letters, the average number of repetitions per message, the number of lowercase letters that appeared before the first capital letter, and the number of times that the word appears in a message by itself. These parameters were represented as columns in the matrix. 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 with which to measure our loss.

After implementing our tests, we found gradient ascent did not work well, and was often less accurate than randomly initializing the weight matrix. On the other hand, gradient descent produced weights that are constantly within a 95% accuracy approximately. Using these trained weights, we obtained a parameter for our clip classifier.

For our clip classifier, we began by formatting the data gathered from the chat log into a matrix. We parsed the data by one-minute timestamps, using the timestamps as the rows of our matrix. The columns of the matrix corresponded to how many messages were in a given 15 second interval, the average length in characters of messages within that minute, 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 period.

Next, we fed the data we had gathered, in the form of an N-by-four list into logistic zero-and-one regression. We first created a randomly-generated weights list in which to hold our outputs, itself also a list of dimension N-by-four. Next, our program ran optimization functions to get an updated set of weights. One of the functions used gradient ascent, while the other used gradient descent. We then ran prediction on both weights, and wrote a function to count loss and compare it with the output.