Video Game Session Detection

Jacky Nie

Aug 2022

1 Introduction

As the revenue for selling hardware continuous to decline for smart TV companies, companies like Vizio are relying on analyzing users' engagement behaviors and advertisement revenue more than ever. One topic about analyzing users' engagement behaviors is to know what video games users are playing through the TV and how long the users are playing those games.

It appears that image recognition is not suitable for detecting video games because image contents in video games are too noisy. Therefore, in this report, I will present the state-of-the-art ways to detect video games with audio fingerprints collected from the users' end.

The contribution in this paper is that it gives a comprehensive model that detects the change-of-point, establishes sessions, and labels each session to a certain game.

The sections are arranged as follows: Section 2 gives an overview of the models used in this paper and the raw data, Section 3 explains how to detect games by similarity matches and how to smooth the decision boundaries, Section 4 briefly covers how to detect a game console sound, Section 5 covers how to detect whether a session of audio fingerprints belong to a game or tv contents, and Section 6 covers how to detect a change-of-point given that the previous three models do not capture such signal. Section 7 gives a summary of the comprehensive model that is based on the previous models, and show some examples.

2 Overview

The models in this paper can be categorized into two groups by their functionalities: 1. detect a change-of-point, 2. assigns a game label to a session.

2.1 Detect Change-of-Points

I assume there are four ways that users can change their behaviors and we can establish game sessions:

1. The user switches from playing Game A to Game B from the home screen.
2. The user switches from playing Game A to Game B without using the home screen.
3. The user switches from playing Game A to watching TV content from the home screen.
4. The user switches from playing Game A to watching TV content without using the home screen.

If either of the above four cases happens, then I will end the current session, and start a new session. The False Negative Rate is the criteria that I are trying to minimize here because creating more sessions than it should be is better than creating less sessions than it should be. If I create more sessions than needed, and catches all the change-of-point, then the total amount of time users spend on playing a particular game does not change much. However, if I catches less change-of-points than there are, then I might be falsely assigning a wrong label to sessions.

2.2 Assign Game Labels to Established Sessions

Once I establish a session by detecting a change-of-point, then I will feed the whole session of fingerprints into our similarity match model to find the right game. Note that I allow for assigning NULL to a session here.

2.3 Audio Fingerprint

The raw data used in this paper are audio fingerprints collected from Vizio smart TV. Each fingerprint summarizes the audio content of a 100 ms session and $X_i \in M^{35}$ where M is an integer from 0 to 255.

3 Similarity Matches and Smoothing

In this section, assume I already establish a session and know that the whole session either belongs to one game only or belongs to a non-game content.

3.1 Approximate Nearest Neighbor

K Nearest Neighbor (KNN), as a similarity-based match algorithm has a time complexity of O(N) and space complexity of O(N*d) where d is the number of labels. When I have a relative large dataset, KNN becomes non-practical to be implement into the production line. Thus, Approximate Nearest Neighbor (ANN) is used instead which trades accuracy for efficiency. More details about ANN can be found on https://faiss.ai/.

The training data for ANN are audio fingerprints of game-playing of 310 popular console games. Each data point is a key-value pair with key $X_i \in M^{35}$ and value $Y_i = 1, 2, 3, ..., 310$. The algorithm is as follows: given an audio fingerprint $X \in M^{35}$, ANN will return the closest K points with labels in the training set and their Euclidean distances to X. If the labels for the most similar matches of X are all from one game 'Game A', and the distances are close enough, then X will be labeled as 'Game A' with high confidence. However, it seems unclear how to choose the decision boundary.

The algorithm this paper adapts is to set K=4 and find the four most similar matches for data X and their distances. First, denote the ground-truth label for X as Y_{True}. Denote the label for the most similar matches as Y_1, Y_2, Y_3, Y_4 and denote the distances to the four matches as D_1, D_2, D_3, D_4 respectively. Then, I assume there exists a function f_{ANN} such that $f_{ANN}(D_1, D_2, D_3, D_4)$ gives the probability that $Y_1 = Y_{True}$.

3.2 Feature Engineering

I found that important features that have predictive power not only include D_1, D_2, D_3, D_4, but also include $\frac{D_4-D_1}{D_1}$, $\frac{D_3-D_1}{D_1}$, $\frac{D_2-D_1}{D_1}$. The last three features can be considered a measure of how far each neighbors are from each other.

3.3 Classification Model

With the features mentioned in the previous part, I use an XGBoost classification/Logistic Regression model to fit and estimate f_{ANN}, denote as f'_{ANN}. The significant features are $\frac{D_4-D_1}{D_1}$ and D_1. It is trivial that D_1 is an important feature. The theory behind $\frac{D_4-D_1}{D_1}$ as an important feature is that sometimes even D_1 is small, Y_1 tend not to be the right label. This is because D_1, D_2, D_3, D_4 are also small and Y_1, Y_2, Y_3, Y_4 are not the same label. This means the data point X belongs to a rather noisy part of a game where multiple games can have the same sound. In this case, $\frac{D_4-D_1}{D_1}$ could be a useful indicator.

3.4 Result

The validation accuracy for ANN is 50.9%. The validation accuracy for XGBoost is 75.2% with a recall of 75.4%, and a precision of 73.5%. The validation accuracy for Logistic Regression is 73.7% with a recall

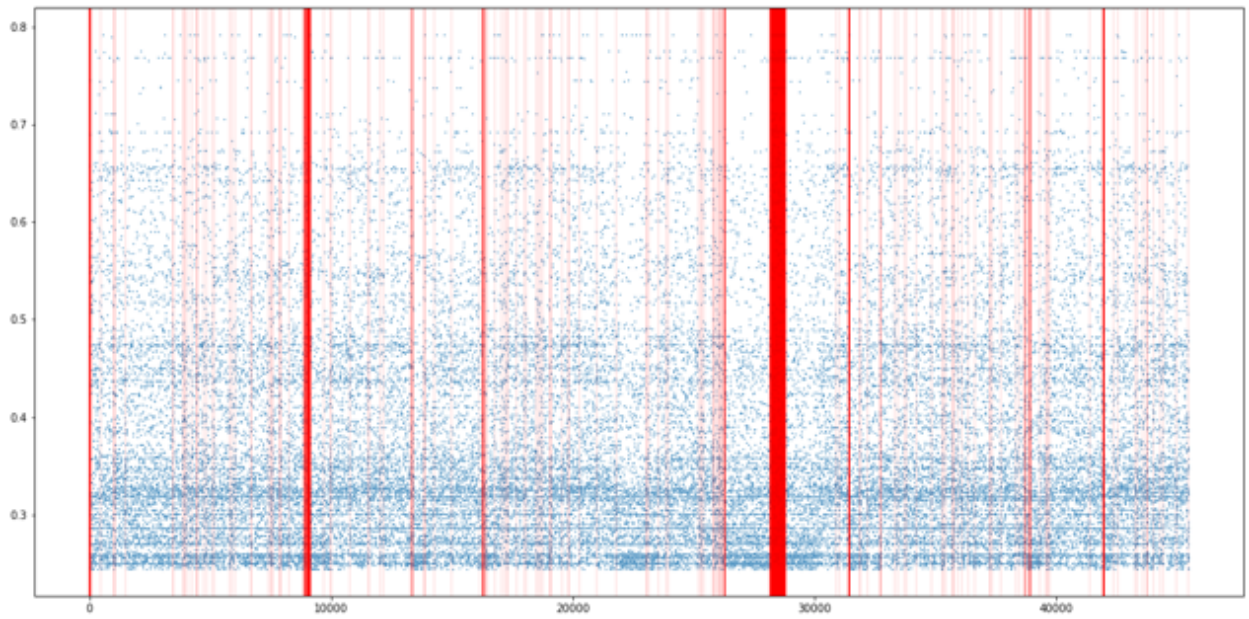

Figure 1: Overwatch Game-Playing Example Before Aggregated Average

of 71.2%, and a precision of 78.3%. In Figure 1, it is about an hour-long Overwatch game-playing. The blue dots are the outputs of f'_{ANN} at each point, which is the chance that the nearest neighbor is the right neighbor. The red vertical lines are when $Y_1 = Y_{True}$. We can see that only a small proportion of the whole session is predicted right, and also that the outputs from f'_{ANN} are noisy.

3.5 Aggregated Average

Therefore, I introduce another layer of smoothing. Given a 200-size window of $Y_{1,i}$s and $f'_{ANN}(\cdot_i)$s: $Y_{1,1}, Y_{1,2}, ..., Y_{1,200}$ and $f'_{ANN}(\cdot_1), f'_{ANN}(\cdot_2), ..., f'_{ANN}(\cdot_{200})$. Also, assume I have a threshold P. The smoothing layer works as follows:

Group by $Y_{1,i}$ and for each unique label Y, compute the average of $f'_{ANN}(\cdot_i)$ for all $f'_{ANN}(\cdot_i)$ such that $f'_{ANN}(\cdot_i) > P$. Then, return the label Y with the maximum aggregated average and the maximum aggregated average itself. If no such Y exists because all $f'_{ANN}(\cdot) <= P$, then return (-1, 0).

We can see from Figure 2 that the decision boundary is much smoother and I detect a small session around fingerprint 30000 correctly with high confidence. Even though only a small part of the fingerprints are detected correctly with high confidence, once we know the whole session belongs to one class, then we can label the whole session by that label.

Figure 3 gives a summary of the model described in this Section.

3.6 Latency

Model	Window Size	Stride Size	Session Length	Time
ANN	NA	NA	40 minutes	79.1 ms ± 1.62 ms
XGBoost	NA	NA	40 minutes	10.6 ms ± 261 µs
Logistic Reg.	NA	NA	40 minutes	3 ms ± 37.6 µs
Aggregated Avg.	200	10	40 minutes	600 ms ± 5.69 ms
Aggregated Avg.	200	50	40 minutes	120 ms ± 1.18 ms

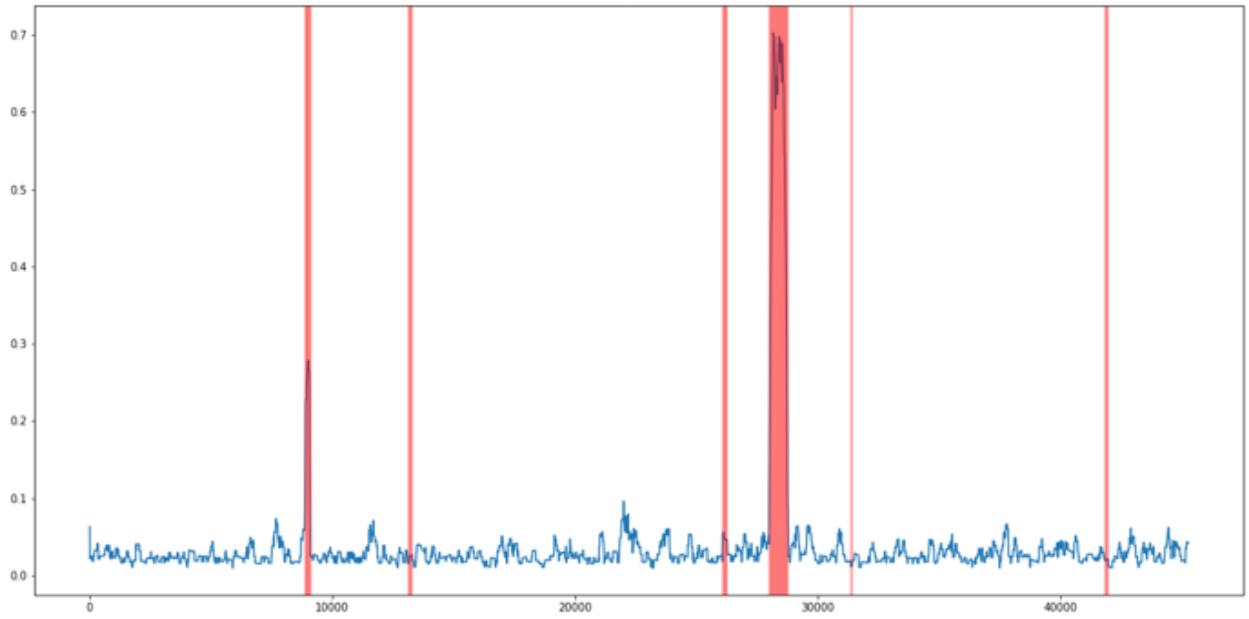

Figure 2: Overwatch Game-Playing Example After Aggregated Average

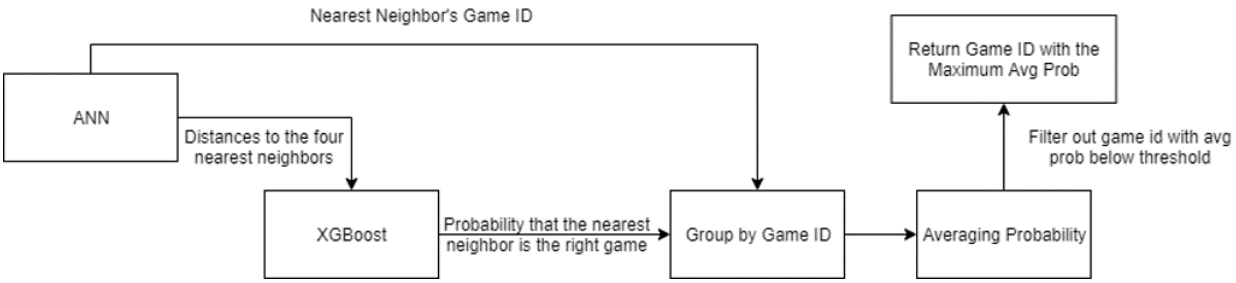

Figure 3: Model Summary for Section 3

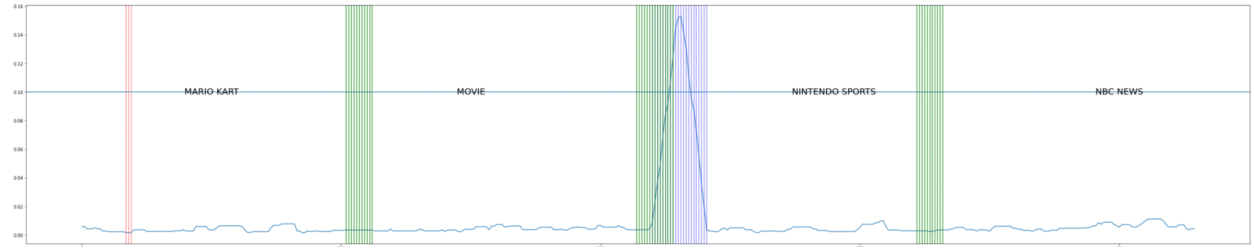

Figure 4: 10-minute Mario Kart, 10-minute movie, 10-minute Nintendo Sports and 10-minute NBC news

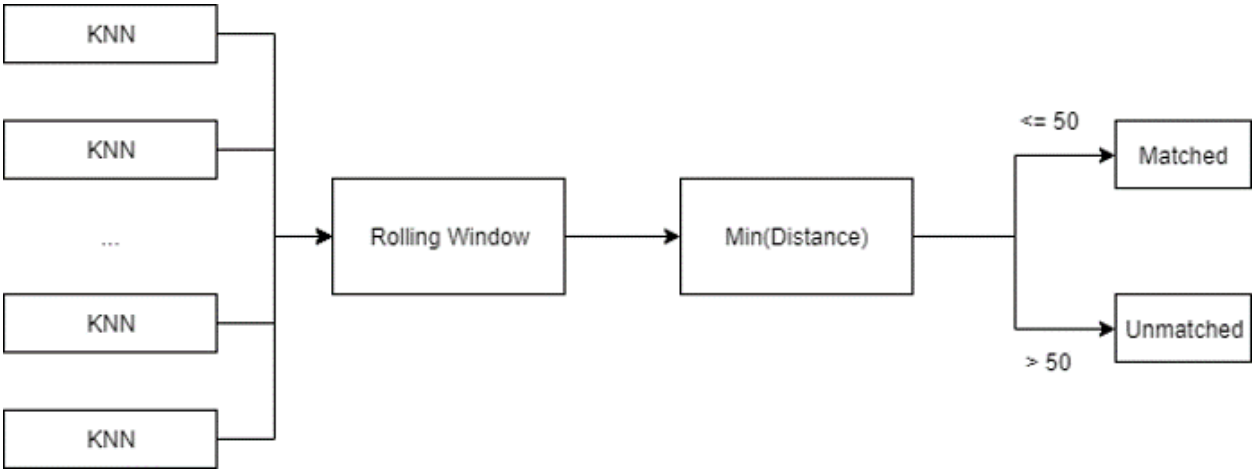

Figure 5: Model Summary of Section 4

4 Home Screen Detection

In Section 3, I assume that I already established a session. This section describes a way of identifying a change-of-point and establishing sessions.

Clearly, if users reach the home screen of their game console, then we can assume with confidence that the users are going to have a different behavior than the previous session. Thus, whenever I detect such action, I shall end and establish the previous session for labelling and start a new session.

The method for detecting a home screen action in this paper is similar to the ANN model, excepts that I use KNN with K=1. This is because game console home screen sound does not occupy too much space and we can drop off the word "Approximate" here.

The algorithm is as follows: for a 50-size window, I find the closest match to a game console home screen sound and thus I will have 50 distances for each of the data point to their closest neighbors. Then, if min(distances) ¡ threshold, I label the window as a home-screen window.

Figure 4 shows an example of a 40-minute session: 10-minute Mario Kart, 10-minute movie, 10-minute Nintendo Sports and 10-minute NBC news. Note that I switch between games and contents through home screen of my Nintendo Switch. The green vertical lines are matches of home-screen and all three of the change-of-points are detected. The blue vertical line represents the output from f'_{ANN} and the purple lines are when the nearest neighbor is the right neighbor for the model from Section 3. We can see that only a part of Nintendo Sports are classified as Nintendo Sports with high likelihood, but we can assign the whole 10 minute session as Nintendo Sports because that is an established session.

5 Game vs No Game Detection

What if users are switching between playing games to watching movie and they do not use the home screen? This could be the next question someone may ask.

Therefore, I then build a model to detect whether a window of audio fingerprints belong to a game or not.

5.1 Customized Transformer

I use a Transformer to handle this task. A Transformer model is a deep learning sequential model. Initially it was designed to take a sequence of words and do natural language processing missions. However, I deleted the embedding layer because our raw audio fingerprints are already embedded in a way. A Transformer is composed of an Encoder and Decoder with Attention and residual connections in between. I only use the Encoder part and add a Max-pool Layer and a Fully-Connected Layer instead for the last SoftMax layer. More details about Transformer model can be read from the famous paper "Attention is all you need".

5.2 Training Detail and Results

The training data comes from both Gameopedia and my self-collected movie/news/show watching data from Vizio's TV. The Transformer will predict the probability that a given sequence of fingerprints belongs to a game.

I used one-layer, 3-head Encoder in this model, and the total number of parameters are 10012, which makes it a related small model in compare to other deep learning structures.

The validation accuracy is 76.5%

5.3 Theories Behind the Black Box Model

The theories behind why this model works is that it detects the frequency of human voices in a given window. If the frequency is high, then it is probably a movie, and vice versa. This is empirically proved when I test the model on a documentary without any human voice, and the model detects it to be a game with high confidence.

5.4 Decision Boundary

The criteria to determine a point to be a change-of-point of switching between games and contents is to when the sessions before this point belongs to a game and the sessions after this point belongs to a moive/show, as shown in Figure 7. Assume the current data point is at index i. Then, I treat fingerprints i-3000 to i as a window, and i+1 to i+3000 as a window. Then, divide each of the two windows into smaller sessions of 500 fingerprints, for about 60 sessions. Feed those 60 sessions into the Transformer, get 60 probabilities that those sessions belong to game, and then average the 60 probabilities out. Without loss of generality, assume the average probability for the first window is greater than 0.5, and that of the second window is smaller than 0.5, then I mark fingerprint i as a change-of-point. If not, then iterate to another fingerprint i+stride

5.5 Latency

Model	CPU/GPU	Window Size	Stride Size	Session Length	Time
Transformer	CPU	500	10	40 minutes	2 s 860 ms
Transformer	GPU	500	10	40 minutes	900 ms

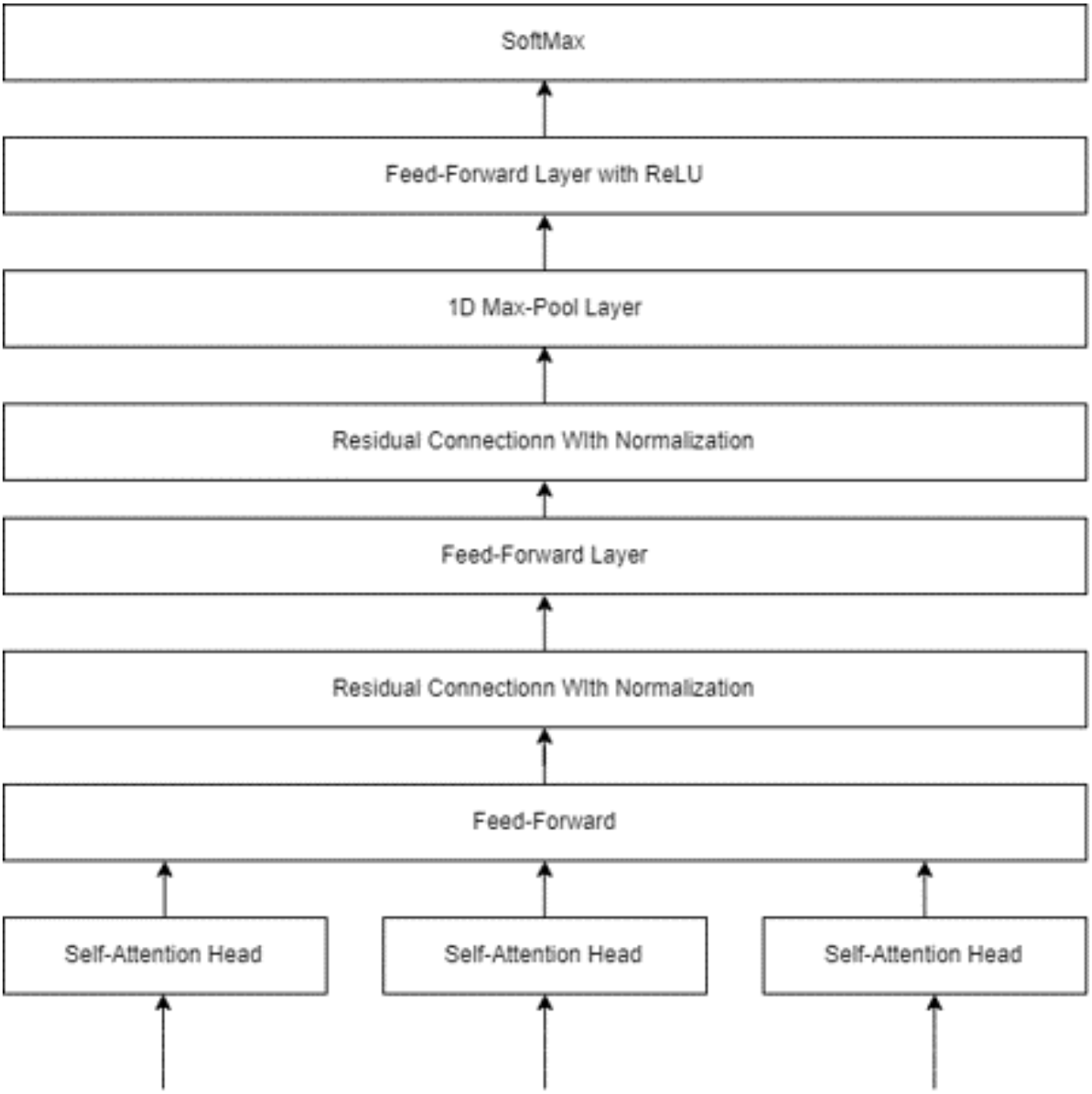

Figure 6: Customized Transformer

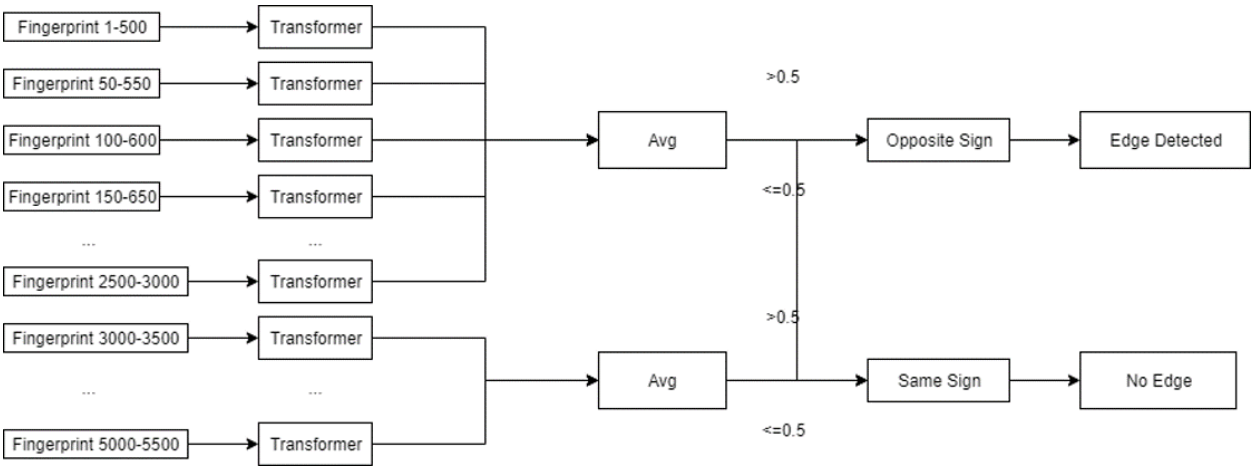

Figure 7: Model Summary of Section 5

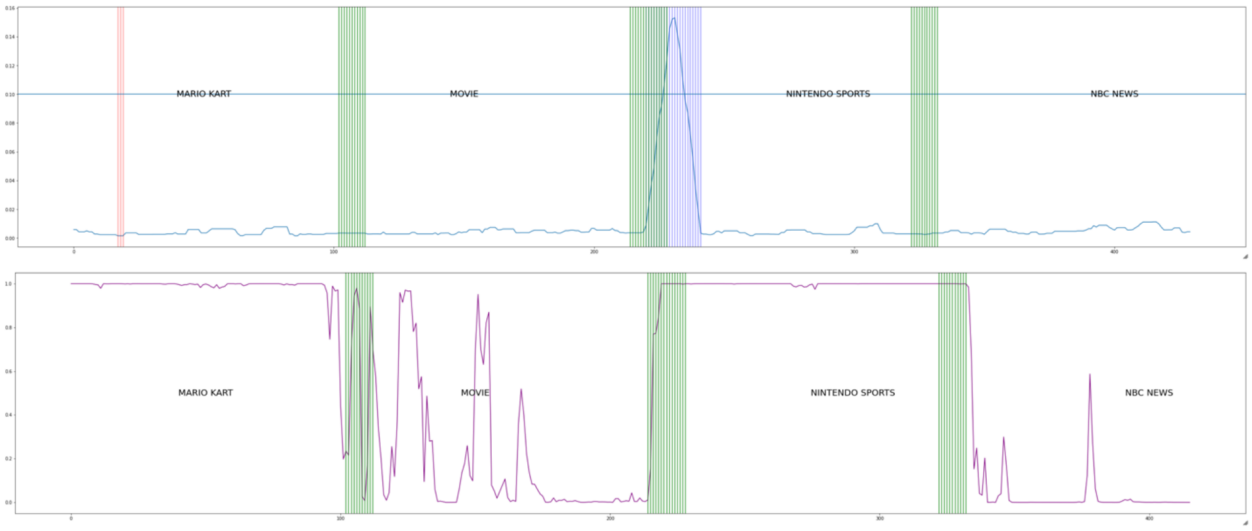

Figure 8: 10-minute Mario Kart, 10-minute movie, 10-minute Nintendo Sports and 10-minute NBC news

5.6 Example

Figure 8 shows the same Mario Kart example as before. The purple lines in the second plot is the probability that a session belongs to a video game or not. We can see that it is constantly close to 1 when I are actually playing games, and then drops to be close to 0 when I switch to movie-watching and news-watching.

6 Chi-Square Distance Based Change-Of-Point Model

Section 4, and 5 both elaborate two cases where a change-of-point can be detected. But it could still happen that a user switches from playing Game A to playing Game B without using the home screen. If this happens, we will need this model to capture the change-of-point.

6.1 Transformer

In this model, I still adapt the usage of Transformer model. But, instead of predicting whether the user is playing game or not, the Transformer in this section focuses on predicting the game label for a session of 50

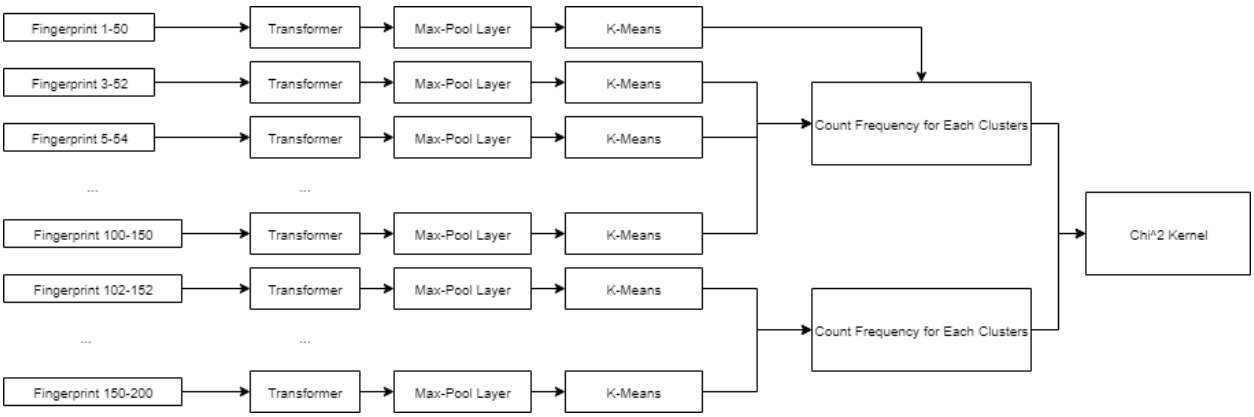

Figure 9: Model Summary of Section 6

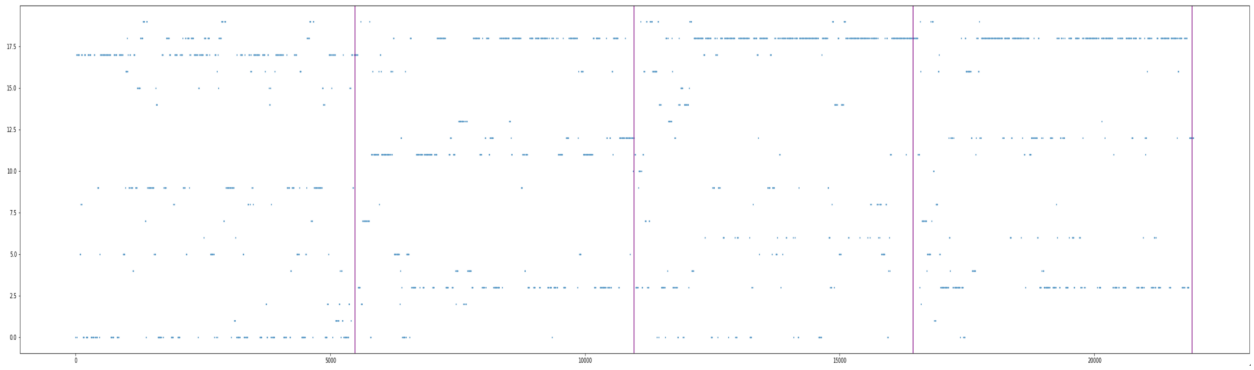

Figure 10: Distribution of Clusters on 10-minute Mario Kart, 10-minute movie, 10-minute Nintendo Sports and 10-minute NBC news

audio fingerprints. The Transformer model here has the same structure as the previous section. However, to handle 310-classification problem than binary classification, the Transformer here has 2 layers instead of 1 layer of Encoder Layers. This increases the number of parameters to 70580. I also filter out game data that is less than 2000 fingerprints. The model reaches a validation accuracy of 49%, which is similar to the Approximate Nearest Neighbor model.

6.2 Max-Pool Layer

Instead of using the last SoftMax layer of the Transformer, I use the Max-Pool Layer as an embedding. After the training, the Max-Pool Layer extracts the locative information in a 50-sized window. The dimension of each embedding from a 50*35 matrix is 35.

6.3 K-Means Clustering

Then, I get the embeddings for all the training data and perform a k-means clustering of 20 centroids.

In Figure 10, I include an example of the distribution of clusters for a 40-minute session consisted of four different contents. We can see clearly that the distribution of the clusters are different for each of the four different contents.

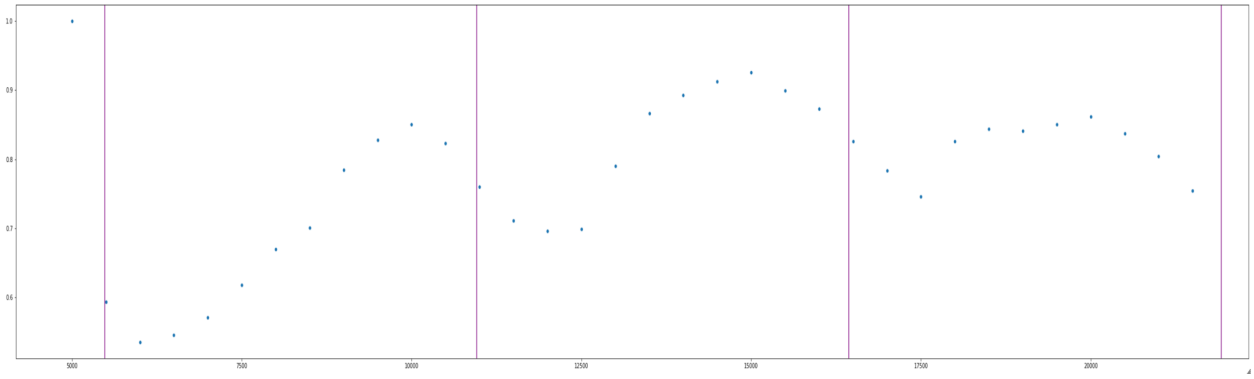

Figure 11: Distribution of Chi-Square Kernels on 10-minute Mario Kart, 10-minute movie, 10-minute Nintendo Sports and 10-minute NBC news

6.4 Histogram-Based Transformation

It is difficult to measure the similarity between two sessions if the data is of the format shown in Figure 10. Also observe the sparsity within the data: each data point can only have one integer value because each data point can only belong to one cluster by definition.

Therefore, I transform the data into histogram-based format. That is, given a window of clusters that data points belong to: [1,2,2,3,1,3,3,3,2,2,2,4], it will be transformed into key-value pairs 1:2, 2:5, 3:4, 4:1 with the key being the cluster index, and the value being the frequency of such cluster index in a given window.

6.5 Chi-Square Kernel

One may immediately realize that the histogram-based format is like a multinomial distribution $X \sim Multinomail(p_1, p_2, ..., p_{20})$ with p_i being the probability that a cluster i appears in each of the position in a given window.

One may also realize that to test whether two multinomial variables come from the same distribution, we need chi-square test. Therefore, I introduce very similar distance metric: chi-square distance. To make it easier, I will describe the chi-square kernel here, which is a inverse monotonic function of chi-square distance: the larger the distance is, the smaller the kernel is.

$$\chi^2 Kernel(X, Y) = exp(-\gamma \sum_{i}^{m} [(X_i - Y_i)^2/(X_i + Y_i)] \tag{1}$$

for some constant γ. Also, note that when both X_i and Y_i are zero, we treat the sum as zero to avoid zero division.

In this model, I set gamma to be 0.01, and Figure 11 shows an example of the chi-square kernel distributions on 10-minute Mario Kart, 10-minute movie, 10-minute Nintendo Sports and 10-minute NBC news.

We can see that when the chi-square kernel time series reach a local minimum with a relatively small number, then it is likely that there is a change-of-point.

Figure 12 shows another example of how the chi-square kernels look like in real data. Notice that the purple lines represent actual change-of-points and the red lines present predicted change-of-points based on chi-square kernel.

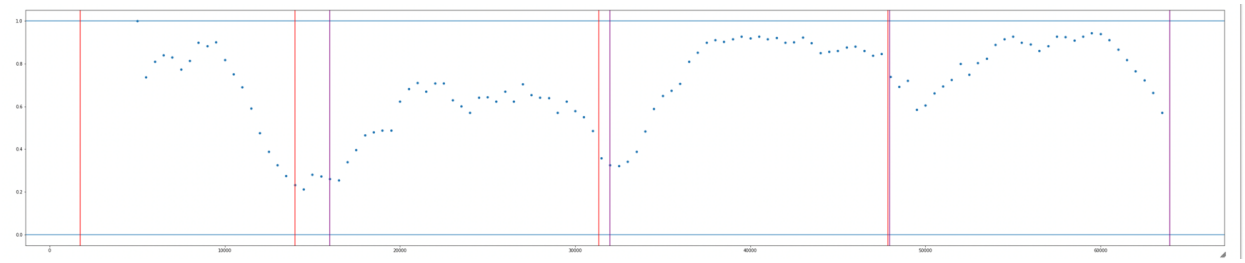

Figure 12: Distribution of Chi-Square Kernels on 30-minute Monster Hunter - Rise, 30-minute movie, 30-minute NBA 2K and 30-minute NBA Streaming

7 Comprehensive Model

We have already seen three models for detecting change-of-point to establish sessions, and one model for labelling established sessions. If we concatenate them together, we have this comprehensive model.

7.1 Change-of-Point Strategy

The strategy for combining three change-of-point-detection models is to mark a change-of-point whenever either one of the three models calls for a change-of-point, and when we mark a change-of-point, we make sure that it is not too close to the previous change-of-point to avoid replication.

7.2 Model Performance

Figure 13 - 17 give 4 examples of the comprehensive model's performance on real audio data. The first plot shows the similarity matches model, the second plot shows the game-vs-no-game model, and the third plot shows the chi-square kernel model. The green vertical lines represent detections of home-screen, the red vertical lines in the first plot represent correct first nearest neighbor from ANN, and the red vertical lines in the third plot represent detected change-of-points while the purple lines in the third plot represent the ground-truth change-of-points.

8 Summary and Next Steps

In summary, the comprehensive model correctly detects most of the change-of-point. But the similarity based labelling model has a relatively poor performance due to data constraint. The immediate next steps would be to collect more fine game-playing data. Then, we can increase the complexity of the model to perform better.

Figure 13: Comprehensive model on 10-minute Mario Kart, 10-minute movie, 10-minute Nintendo Sports and 10-minute NBC news

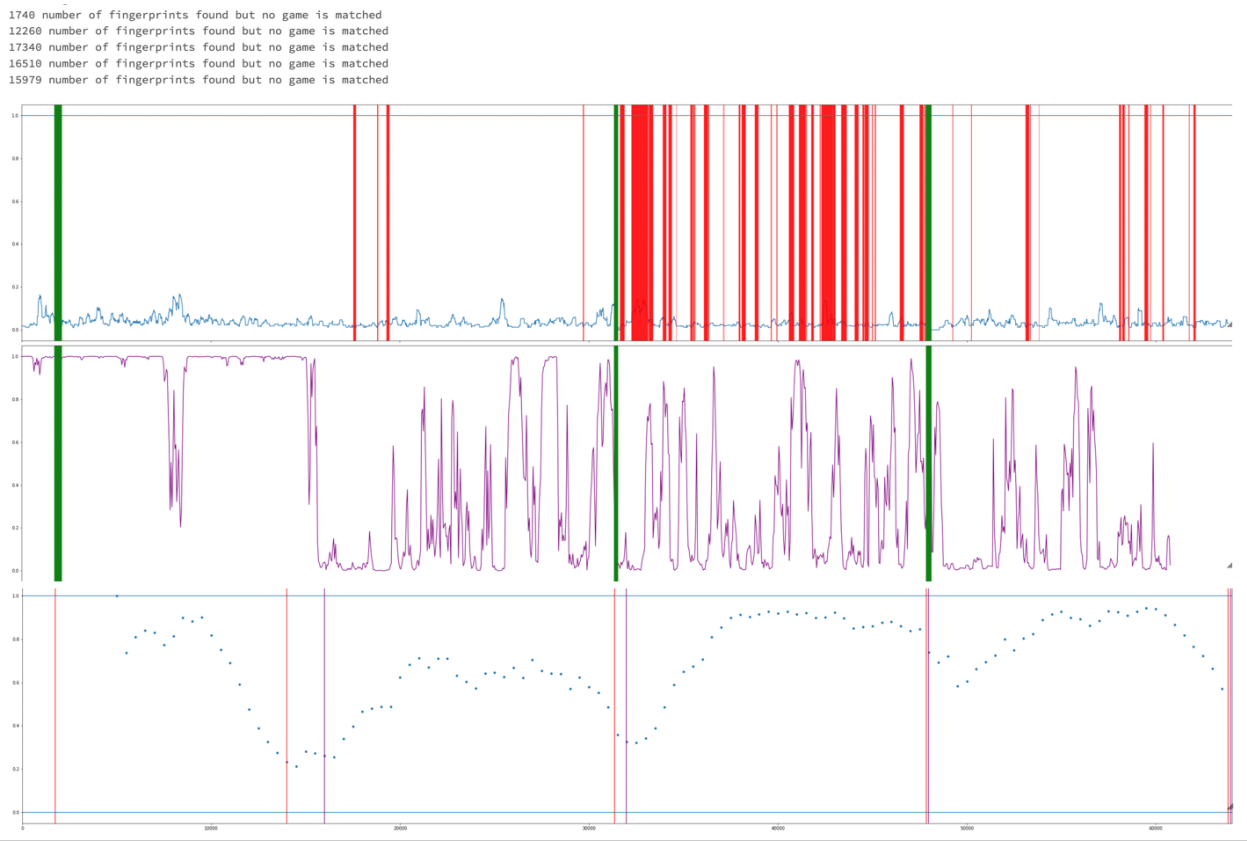

Figure 14: Comprehensive model on 30-minute Monster Hunter - Rise, 30-minute movie, 30-minute NBA 2K and 30-minute NBA Streaming

Figure 15: Comprehensive model on Grid Legend game playing

Figure 16: Comprehensive model on Overwatch game playing