Group 69: Machine learning and sensor data - Audience Affective Response Prediction

Manish Agarwala (2818654), Shourya Rawat (2812517), and Stijn de Groot (2556107)

Vrije Universiteit Amsterdam

Abstract. Current research investigates if machine learning models can accurately predict audience affective responses from smartphone sensor data collected at the end of a performance.

Keywords: Machine learning · Sensory data · Deep Learning

1 Introduction

Audience engagement and satisfaction are critical metrics for evaluating the success of live performances, including concerts, theater productions, and public speaking events. Traditionally, the assessment of audience response to live performances have relied on subjective observations and post-event surveys. These methods, while useful, are inherently limited in their scope and accuracy, since they may not accurately capture real-time emotional and physical reactions of the audience members. The advent of machine learning, coupled with advanced sensor technology, present a feasible approach to objectively quantify audience engagement using objective, real-time data. In this report, we propose a machine learning model that utilizes multimodal sensor data to predict audience response during live performances. Specifically, the model utilizes audio sensors to capture sound levels generated by audience reactions like cheering and clapping, accelerometer and gyrometer to monitor physical movements indicative of engagement, such as standing ovations and rhythmic clapping which often indicate strong emotional reactions and collective enthusiasm from the audience. By combining these data sources, the proposed model can identify patterns and predict levels of audience satisfaction and engagement with a high degree of precision. We also aim to demonstrate the practicality and efficacy of such a model in real-world scenarios.

Live performances such as theatrical productions, musical concerts, and sports events, thrive on audience engagement. The energy and feedback from the audience play a crucial role in shaping the dynamics of the performance. For performers, real-time feedback on audience engagement can provide actionable insights, allowing them to adapt their performances dynamically to enhance connection with the audience. For instance, musicians can modify their setlists or alter their stage presence based on the immediate feedback, a speaker can modify

their pacing or content based on detected audience engagement levels. Moreover, the model can aid in post-event analysis, providing detailed insights into which segments of a performance elicited the most significant responses. This can be invaluable for refining future performances and marketing strategies, as well as help the artists and event organizers understand their audiences better.

1.1 Research Question

The research question that we aim to answer with our machine learning model is as followed:

How accurately can a machine learning model predict audience engagement and satisfaction during a performance using multi-modal sensor data, such as accelerometer and gyroscope data indicating physical movement and audio levels to classify the level of audience engagement.

From a broader perspective, demonstrating the capability of artificial intelligence to interpret human emotions and behaviors in real-time. This model exemplifies the potential of AI to enhance human experiences and provides deeper insights into collective human behavior, thereby advancing the understanding of audience dynamics in live performance settings.

1.2 Dataset Description

The dataset has been collected by the aggregated efforts of the members of our group. The data has been collected using phyphox version 1.1.16 by creating a custom experiment. To create the custom experiment we used the phyphox editor. The experiment uses four sensors, namely a linear accelerometer to record the acceleration on the three planes (x,y and z plane), a gyroscope to quantify the rotational movement on the three planes (x,y and z plane), a barometer to record the pressure in hectopascal (hPa) and a audio sensor to record the sound pressure level (SPL) in decibles. The accelerometer is employed to measure the velocity of movements, thereby indicating the clapping frequency, which assists in determining audience engagement. The gyroscope measures the device's orientation to more accurately capture the micro-movements made by individuals in the audience. The audio sensor measures sound pressure levels, with louder applause signaling a more favorable response to the performance. The barometer records atmospheric pressure within the environment, where fluctuations in pressure indicate changes in altitude, such as audience members standing for an ovation, further reflecting a positive reaction to the performance.

Measurements used
Our experiment uses accelerometer and gyroscope in the three planes (x,y and z) with a frequency of 100Hz and 8000Hz respectively to detect the movements

of an individual, a barometer to record the pressure exerted with a frequency of 8000Hz and an audiosensor that records the raw audio at with a frequency of 100Hz.

Prediction
The developed machine learning model predicts a variable "label". The variable is a categorical variable that represents the intensity of the audience's satisfaction and engagement based on the data from the sensors with four levels, noa (standing for no applause), light, medium and loud. For example, instances that record a high value for the variable "audio" (which represents the volume of the audio recorded in decibels) have a higher label for the output variable since we can say that there is a correlation between the intensity of applause with an individual's satisfaction. Similarly, instances with a high value for the variable "acc_y" and "acc_abs" indicate that an individual made vigorous movements that indicate intense applause and/or a standing ovation and thus subsequently have a higher label for the predicted variable.

2 Dataset Description

2.1 Data summary

The final dataset used to train the models has been compiled through the collaborative endeavors of our group members. The dataset was acquired through Phyphox version 1.1.16 by utilizing a custom experiment developed by us. The experiment employs four different sensors to record various measurements that indicate the response of a person to a performance, such as live theatre or a concert. Specifically, a linear accelerometer is employed to measure acceleration across the three orthogonal planes (x, y, and z), a gyroscope was utilized to quantify rotational movements on the same axes, a barometer was used to measure atmospheric pressure in hectopascals (hPa), and an audio sensor was implemented to record sound pressure levels (SPL) in decibels. The dataset comprises of 120 instances in total, with 30 instances of each level of audience affection.

2.2 Raw Measures

1. Acceleration: in axes x, y, z at 100 hz to detect movements like standing up, hand gestures, and overall activity.
2. Gyroscope: in axes x, y, z at <8000> hz to measure rotational movements of the wrist or phone orientation changes.
3. Audio: raw audio at <100 hz.

2.3 Extracted Measures

1. Audio amplitude: strength of sounds.
2. Frequency spectrum: freq spec.

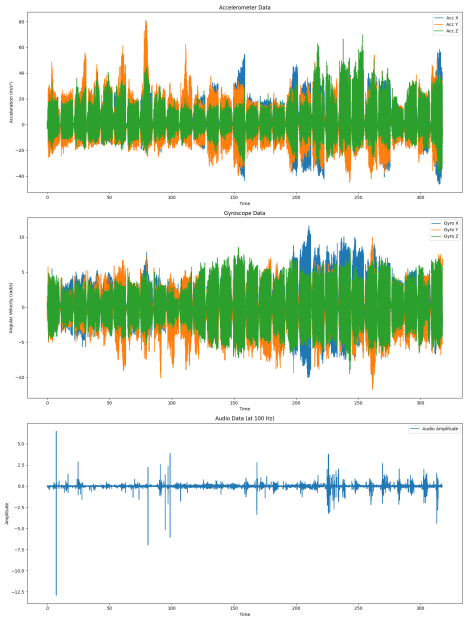

Fig. 1. Plots of time against processed Acceleometer, gyroscope and audio data

3 Handling Sensory Noise

The collected dataset initially contains a lot of missing values. The missing values could be caused by the sensors not providing the measurements aa certain points in time, which could occur in case the sensors record data at different intervals. We observe this in our dataset as the audio sensor records data at a different interval than the other sensors. We broadly used imputations to deal with missing values.

3.1 Handling Missing Values

After outlier detection and removal using chauvenet's criterion, we observe that the dataset now contains some missing values. To deal with these observed missing or NaN values, we used the interpolation method. Interpolation is a powerful approach to fill in missing values in time-series data by estimating the values based on the data points before and after the missing timestamp. We chose imputation by interpolation as it is particularly useful for time-series data, where mean imputation techniques may not be suitable. We used the default parameters of linear interpolation since this method is quick and effective, especially when the missing data follows a known pattern. Subsequently, all the columns that contain values taken from the accelerometer and the gyroscope have been imputed by interpolation. For the column 'pressure', we decided not to follow the

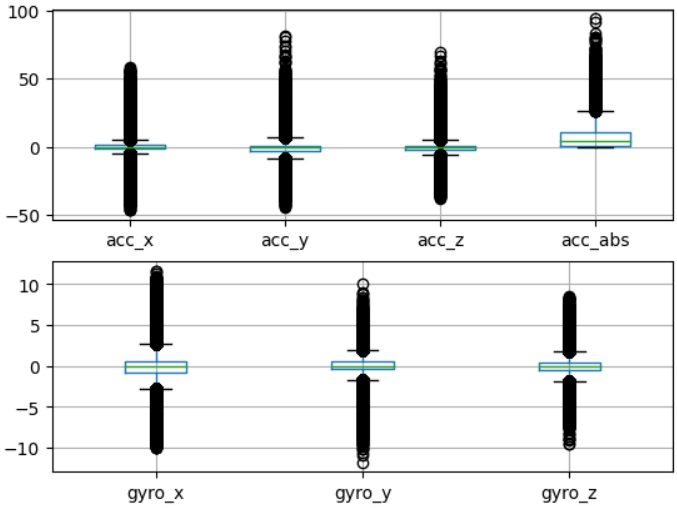

Fig. 2. Box Plot of Accelerometer & Gyroscope data

interpolation approach and instead opted for simple imputation using forward fill, which assigns the last known value for pressure to a missing value.

3.2 Handling Noise

In our data cleaning section, We focus on removing noise and handling missing values. First, We use Chauvenet's criterion to detect and remove outlines. It identifies data points that significantly deviate from the normal distribution, Also it ensures that extreme values do not skew the analysis. Next, We handle the missing values in our dataset. For this we use linear interpolation. It predicts missing data points based on Other values and providing more accurate imputations.

Finally, using the Butterworth lowpass filter to smooth the data point also reduces high-frequency noise. Moreover, This process can effectively predict random fluctuations and keep the essential trends in the dataset. Using those techniques ensures the dataset's quality, consistency, and it is clean.

4 Feature engineering

Feature engineering is a very significant step in machine learning. In our sensory data type feature engineering is even more important. According to Géron, A., Feature engineering is crucial because it directly impacts the predictive performance of machine learning models. He also claims that it transforms raw data into meaningful features, it helps models capture the underlying patterns and relationships more effectively [2]. As we know, In our dataset we have various

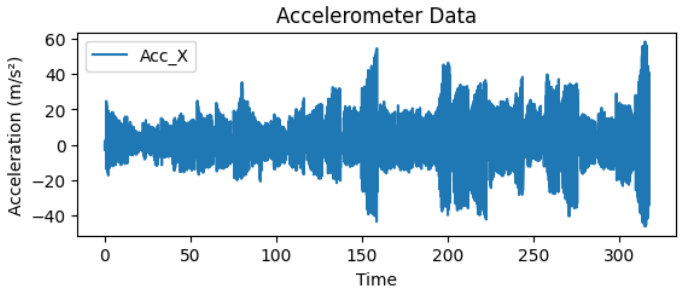

Fig. 3. Filtered Accelerometer data after applying Butterworth lowpass filter for frequencies above 1.5Hz

sensory inputs such as accelerometer data. Here, Our goal is to transform raw sensory data into meaningful features which can be used to predict our result more certainly.

4.1 Feature Engineering Steps

Time Domain Features
At first, we look into time domain features derived directly from our raw time-series data. We have engineered the following features:

1. Statistical Measures:
We calculate the mean by averaging the time window to 10 seconds. Also, we use the standard deviation To measure the data dispersion around the mean and lastly, We fix the value to a certain range of the data.

2. Window Aggregation:
We aggregated features over sliding windows to capture local trends and patterns. We experimented with various windows of sizes, For example, 5 - 40 second windows.

3. Activity Recognition Features:

Next, We calculated the Root Mean Square (RMS) of the acceleration data to capture movement intensity which helped our model significantly. We also wanted to measure the activity level. We summed up the squares of sensor values within a window.
4. Derived Metrics:
We compare the rate of change in acceleration, which shows sudden movements, and use GPS data to understand mobility patterns.

Frequency Domain Features

Next, we work with frequency domain features to measure the periodic patterns and signal characteristics. We have engineered the following features:

1. Fourier Transformations:
We use it in accelerometer and gyroscope data to make it a frequency domain where we get features such as dominant frequency and spectral entropy.

2. Wavelet Transforms:
With the help of Wavelet Transformation, we can access the data at multiple scales and calculate the transient patterns, which helps us collect features as coefficients from different wavelet levels.

3. Power Spectral Density (PSD):
We applied PSD to measure the signal's power distribution.

5 Application of Machine Learning Algorithm

5.1 Train-Test Setup

In the context of classical machine learning, splitting a dataset into training and testing sets is a critical step to ensure the robustness and generalizability of the predictive models. For this study, we utilize a sensory dataset comprising various sensor measurements such as accelerometer, gyroscope, and pressure data, with the inclusion of a categorical target variable called 'label'. The objective is to predict the variable 'label' based on the provided sensor readings using a random forest model. Each row in the dataset represents a set of sensor readings at a specific timestamp, with columns indicating different sensor modalities and the target variable. The primary feature columns, excluding the target column, constitute the input variables, while the target column serves as the output variable.To ensure an unbiased evaluation of the machine learning model, the dataset is partitioned into training and test sets using the 'train_test_split' function from the 'scikit-learn' library. This function facilitates a random allocation of data, which mitigates the risk of selection bias. For this specific division, 80% of the data is allocated to the training set, while the remaining 20% is designated for the test set. This stratification ratio is chosen based on standard practices in machine learning literature, balancing the need for sufficient training data to build an accurate model and enough test data to evaluate its performance reliably
The training set is utilized to train the random forest model. During this phase, the model learns the underlying patterns and relationships between the input features and the target variable. The learning process involves constructing decision rules that partition the feature space in a way that optimally predicts the target labels. Post-training, the model's performance is assessed on the test set . This methodology ensures that the model is not only tailored to the specifics of

the training data but also exhibits robust performance when applied to new, unseen data. The adherence to these established protocols enhances the credibility and applicability of the model in practical scenarios, where predictive accuracy and generalization are paramount.

5.2 Training a Machine Leaning Model

After expensive evaluation of our particular classification problem, we decided to train our predictive model using random forest. With the data partitioned, the random forest model is instantiated using the 'RandomForestClassifier' class from 'scikit-learn'. The model is trained using the default settings to establish baseline performances. After training, the model is evaluated on the testing set. The evaluation metrics included accuracy, precision, recall, and F1-score, calculated for each class and averaged across classes. These metrics provide insights into the model's capability to correctly classify instances and its effectiveness in handling imbalanced datasets. This initial evaluation highlighted the performance gaps and potential areas for improvement in model predictions.

Hyperparameter Optimization
To enhance model performance, a grid search was performed using the 'GridSearchCV' class from 'scikit-learn' framework. It exhaustively explores different combinations of hyperparameters to find the best set for a given machine learning model. The grid search aims to find the optimal hyperparameters by exploring a predefined space of possible values and evaluating model performance using 5-fold cross-validation.

```
Best set of hyperparameters:  {'alpha': 0, 'learning_rate': 0.5, 'max_depth': 7, 'n_estimators': 50}
Best score: 0.8275972675972675
```

Fig. 4. Hyperparameters found using GridSearchCV

Using the best parameters identified from the grid search, the model was retrained on the entire training dataset. The retrained model was then used to make predictions on the test dataset. The final step involved evaluating the optimized model on the test dataset using the same evaluation metrics as in the initial evaluation. This allowed for a comparison between the baseline and optimized model to assess the effectiveness of the hyperparameter tuning.

Results
We have used the Random Forest classifier to predict the performance of our dataset and it shows promising results. Initially, the model was predicted with an accuracy of 0.70, which shows our model and dataset need improvement.

So, With important feature selection, standardization, and dimensionality reduction our classifier's performance improved significantly. However, Hyperparameter tuning helps a lot to optimize the model and achieve an accuracy of 0.82. Finally, We realize the importance of preprocessing and fine-tuning in machine learning, with the right adjustments we can achieve higher accuracy on simpler models. Our improved model gives a balanced classification of multiple performing categories also showing robustness.

```
Accuracy: 0.7
Classification Report:
              precision    recall  f1-score   support

           0       0.75      0.79      0.77        19
           1       0.60      0.55      0.57        11

    accuracy                           0.70        30
   macro avg       0.68      0.67      0.67        30
weighted avg       0.70      0.70      0.70        30

Confusion Matrix:
[[15  4]
 [ 5  6]]
```

Fig. 5. Random Forest classifier

6 Application of Deep Learning Model

6.1 LSTM

Long Short Term Memory networks (LSTM) are a form of recurrent neural networks. They are designed to capture temporal dependencies in sequential data. Currently, we implemented an LSTM model to classify audience engagement levels from (multimodal) sensor data.

6.2 Data Preparation

Steps taken to prepare the data are:

- **Splitting the Data:** The dataset is split into training, validation, and test sets using a stratified split based on recording IDs to ensure that each set contains a balanced distribution of labels.
- **Standardization:** The features (accelerometer, gyroscope, pressure, and audio) are standardized using the StandardScaler to make sure that each feature has a mean 0 and a standard deviation of 1.
- **Reshaping:** We reshaped the data into 10 second recordings, to make sure that each recording has exactly 1000 time steps (data is sampled at 100 Hz).
- **Conversion to Tensors:** The reshaped data is converted into PyTorch tensors, which are required for training the LSTM model.
- **Dataset and DataLoader Creation:** Custom Dataset objects are created for the training, validation, and test sets. Data loaders are then created to help batch processing during model training and evaluation.

6.3 Hyperparameter Settings

Hyperparameters used in our model are:

- **Input Size:** The number of features of the input data. For our dataset, this corresponds to 9 features (accelerometer data in three axes, gyroscope data in three axes, pressure, and audio). So here we chose to use only the sensor data and not apply feature engineering.
- **Hidden Size:** Amount of neurons in the LSTM layer(s). We set the hidden size to 64, because it balances the model's ability to learn (complex) patterns with computational efficiency.
- **Number of Layers:** Amount of stacked LSTM layers in the model. Currently we use 2 layers. This allows the model to capture more complex temporal dependencies.
- **Dropout Rate:** Dropout is implemented to prevent overfitting, and create solutions that generalize better. A dropout rate of 0.5 was used.
- **Batch Size:** The number of samples per batch during training. We chose a small batch size of 6 due to the limited size of our dataset. This is because our current dataset is not that big.
- **Learning Rate:** The step size at each iteration to change it's parameters, while moving toward a minimum of a loss function. We used a learning rate of 0.001.
- **Number of Epochs:** The number of complete passes through the training dataset. We trained the model for 15 epochs, which was sufficient for convergence.

6.4 Model Training and Evaluation

Currently we trained with the Adam optimizer and Cross-Entropy Loss function. The training process:

1. **Forward Pass:** Pass the input data through the LSTM layers and a fully connected layer.
2. **Loss Calculation:** The loss between the predicted and true labels is calculated using the Cross-Entropy Loss function. This loss function converts the logits to probabilites through a softmax, takes the logarithm to get the log probabilites and applies a negative log likelihood to get to a final loss value.
3. **Backward Pass:** Next, gradients are calculated through backpropagation.
4. **Parameter Update:** The parameters are updated using the Adam optimizer.

The model's performance is evaluated on the validation set after each epoch, and metrics such as accuracy, precision, recall, and F1-score are recorded. The final performance of the model is evaluated on the test set, and the results are visualized using confusion matrices and classification reports.

6.5 Results

The LSTM was trained for 15 epochs. Performance was evaluated using metrics: accuracy, loss, f1-score, precision, recall and a confusion matrix.

Training and Validation metrics Figure 6 shows the training and validation accuracy per epoch. The initial accuracy starts at roughly 36% which is a bit higher than random guessing (4 classes would give 25 %). We see the model is indeed learning as the accuracy (generally)increases each epoch.We reach a final 98.61% accuracy for the training data and 95.83 % validation accuracy. This shows that the model is able to learn from the data well and that it seems to generalize. Noteworthy is that around epoch 10 and 11 we reach a validation accuracy of 100 %. This might suggest that the overall data is might not be diverse enough and that the validation set could be too small. We see a similar pattern for the loss. The general trend of the loss is decreasing consistently over the epochs for both the training and validation data. Noteworthy is an upwards spike in the loss around epoch 4. This could be explained by batch variability, such that the model got a batch that was harder to learn from than other batches, especially considering the low batch size. Or it could have adjusted to a new region in the loss landscape. However, we see that the model recovers and continues it's descent across the loss in the following epochs. A final observation is that for our current problem and set up the training loss continues to decrease while the validation loss seems to slowly start increasing around epoch 11. This could be early signs over overfitting.

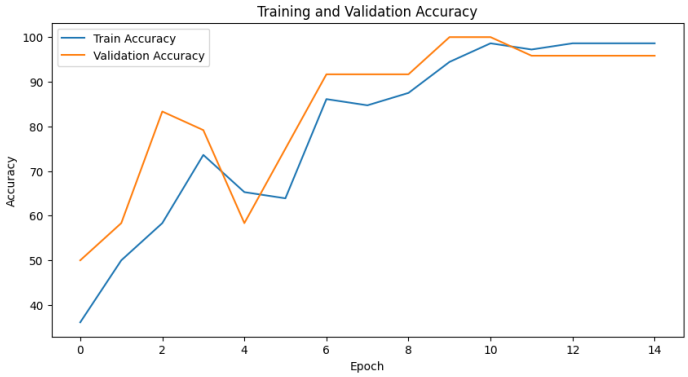

Fig. 6. Training and Validation Accuracy

Test Accuracy and Confusion Matrix After training, the model performance was evaluated on the test set. Resulting in a test accuracy of 92.00%. This indicates that the model maintained high performance on unseen data.

Fig. 7. Training and Validation Loss

Figure 8 displays the confusion matrix for the test set. The model correctly classified all instances of noa (no applause) and light applause with perfect precision and recall. For the medium and loud, the model showed some mis-classifications: 2 time series of loud were classified as medium. Despite these mis-classifications, the overall performance showed robust across all classes.

Table 1. Classification Report for Test Set

Class	Precision	Recall	F1-Score	Support
noa	1.00	1.00	1.00	6
light	1.00	1.00	1.00	6
medium	0.75	1.00	0.86	6
loud	1.00	0.67	0.80	6
accuracy			0.92	24
macro avg	0.94	0.92	0.91	24
weighted avg	0.94	0.92	0.91	24

Classification Report Table 1 shows the precision, recall, and F1-score for each class. The noa and light classes achieved a perfect score across all metrics. This indicates that all classifications from those classes were indeed correct and that all instances from those classes were indentified. TThe medium class showed a precision of .75. This indicates that 75 % of the instances that were predicted as medium were correct. The recall of 1 shows that all the actual medium instances were correctly identified. For the loud class we found a precision of 1, which shows that all instances that were classified as loud were indeed correct. However, we observe and recall of .67. This indicates that 67% of the actual loud instances were correctly identified. The weighted average precision, recall, and F1-score were .94, .92, and .91, respectively. This shows a single score for the model

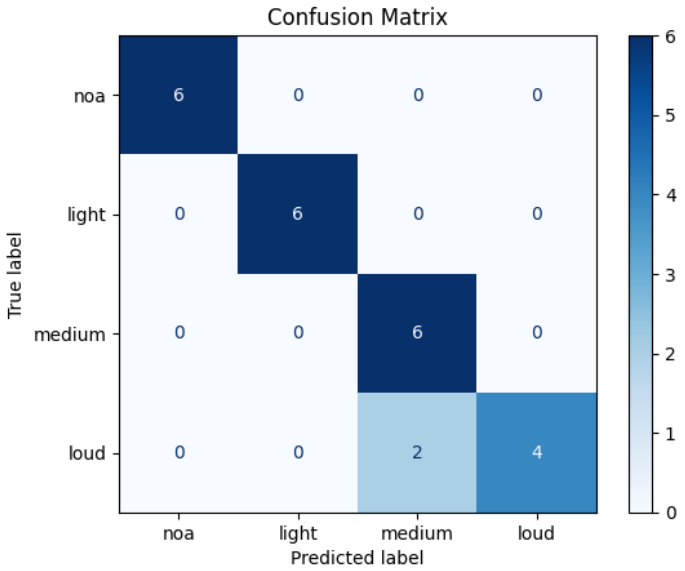

Fig. 8. Confusion Matrix for Test Set

performance across all classes. This is weighted by the number of instances in each classs, however due to the small dataset we used a stratified and balanced set up. We see that the overall accuracy is 92%. So on average the model predicts 92 % correct. A weighted average recall of .92 shows that, on average, 92% of the actual instances were indeed correctly predicted. The weighted average F1 score of .91 indicates balanced performance across all the classes.

Summary of Results The results show that the LSTM model can effectively capture the temporal dependencies in the sensor data. It achieves high accuracy on both the validation and test sets. The model showed good performance in predicting noa and light applause classes, and minor mis - classifications in the medium and loud classes. The decrease in loss and the high accuracy metrics demonstrate the model's robustness and reliability for predicting audience engagement levels based on sensor data. Slight overfitting is observed from epoch 11 and onward.

7 Discussion

Current research aimed to investigate the effectiveness of machine learning model in prediction of audience effective responses from multimodal sensor data. The sensor data was used to classify audience engagement levels into four categories: no applause (noa), light, medium and loud. Both a Random Forest and LSTM

model were applied to try and accurately classify audience responses. Both models showed that it can be applied to effectively capture nuances in the sensor data and gain insight into audience engagement.

The LSTM model effectively captured the temporal patterns in the sensor data and achieved high performance scores across all metrics. The model showed the ability to generalize well to unseen data. However, some mis-classifications were observed where instances that were loud were labeled as medium. This was to be expected as these conditions are very similar with mainly a difference in magnitude of the human behaviors. And with variation within the classes, where in reality some measures in the classes could overlap in the extremes.

Current research demonstrates that machine learning models can effectively predict audience engagement and satisfaction after or during a performance using sensor data. The current research has some limitations. The data was collected in a controlled environment, which does not fully replicate a natural setting with it's complexities. For example, loud music or neighbouring audience members giving loud applause could introduce noise and make it more challenging to correctly classify the data. Moreover, a wide range of human behaviors might not be present in our controlled setting which might occur in a natural setting. More specific, because we have a limited amount of participants, we might not capture a lot of different expressions of engagement and satisfaction.

Despite these challenges, current research provides a foundation for future research to investigate these methods in a more natural setting. And allow for machine learning models that are applicable in real-world scenarios.

References

1. Adami, A., Disch, S., Steba G., Herre J.: Assessing applause density perception using synthesized layered applause signals. In: Proceedings of the 19th International Conference on Digital Audio Effects (DAFx-16), pages 183–189, Brno, Czech Republic, 2016.
2. Géron, A. (2019). "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems." O'Reilly Media.
3. How to filter noise with a lowpass filter - Medium, https://medium.com/analytics-vidhya/how-to-filter-noise-with-a-low-pass-filter-python-885223e5e9b7.
4. Fourier Transformation - Towards Data Science, https://towardsdatascience.com/fourier-transform-the-practical-python-implementation-acdd32f1b96a.
5. Human Activity Recognition with Smartphones - Kaggle, https://www.kaggle.com/code/ingbiodanielh/time-series-classification.
6. Random Forest Hyperparameter Tuning in Python, GeeksForGeeks https://www.geeksforgeeks.org/random-forest-hyperparameter-tuning-in-python/.
7. Evaluating a random forest model, Medium https://medium.com/analytics-vidhya/evaluating-a-random-forest-model-9d165595ad56.
8. LSTM for time series prediction, Machine Learning Mastery https://machinelearningmastery.com/lstm-for-time-series-prediction-in-pytorch/.