Classifying Engagement States Using Wearable-Compatible Biometrics

Ira Kulkarni

1. Motivation

Modern life is saturated with distractions that impair our ability to stay focused, and the consequences of this are well documented: studies show that people are unfocused nearly half the time. These cognitive lapses not only hinder learning and performance but also reduce productivity at scale, affecting both individuals and organizations. Given this, I was interested in exploring how physiological and environmental signals could be leveraged to identify states of engagement using data compatible with wearable sensors.

While prior studies have explored mental state classification using EEG or other complex modalities, my approach is unique in prioritizing real-world feasibility. I focused on variables that are already tracked by common wearables (such as heart rate and movement), aiming to build a passive, scalable solution. The growing accessibility of biometric wearables and the public interest in mental performance optimization (e.g., Pomodoro methods, productivity apps) made this project both timely and practically valuable.

2. Problem Definition

The core question I sought to answer was: *Can we classify a person's cognitive engagement state using only physiological and environmental data compatible with wearables?* I translated this into a supervised multi-class classification problem, where the target variable was the Engagement Level (Low, Moderate, High) and the predictors were five continuous features (HeartRate, Skin Conductance, Temperature, Activity Level, Ambient Noise Level) collected in the Emotional Monitoring Dataset.

This problem addresses a growing need in behavioral health to develop passive methods for real-time attention tracking. If such models are accurate and interpretable, they can be embedded into wearables to help users identify periods of peak focus or suggest break times.

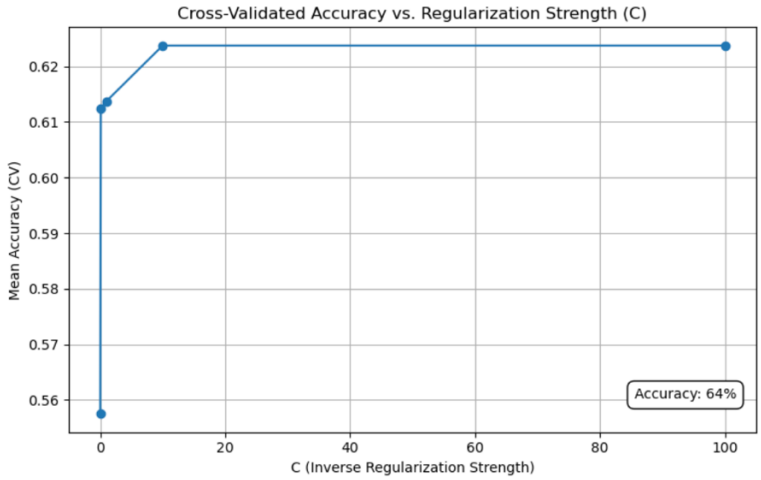
3. Methods And Results

3.1. Logistic Regression

To establish a baseline for engagement-level classification, I implemented a multinomial Logistic Regression model using Python's `scikit-learn` library. Logistic Regression is a linear classifier that offers simplicity and interpretability, making it a natural starting point for supervised learning tasks.

Initially, I used default hyperparameters and then refined the model using `GridSearchCV` with 5-fold cross-validation to optimize the inverse regularization strength `C` and the solver (`'lbfgs'` and `'saga'`). Despite tuning, there was no meaningful improvement in performance—accuracy remained stable at around 64%.

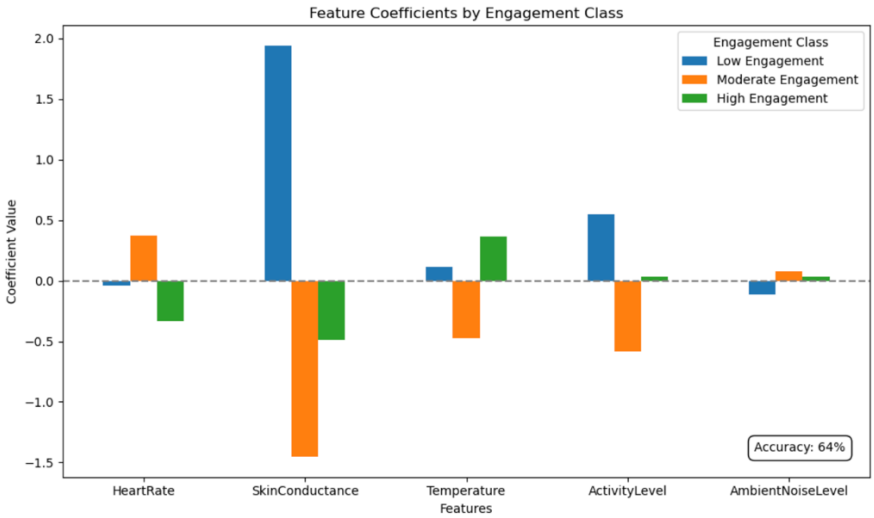
Figure 1: Cross-validated accuracy vs. regularization strength `C`.



The curve clearly plateaus around `C = 10`, confirming that additional regularization adjustments did not meaningfully affect the model's ability to generalize. This supports the idea that Logistic Regression's core limitation here was its assumption of linear separability between engagement levels.

To better understand the model's behavior, I visualized the learned feature coefficients for each class.

Figure 2: Feature coefficients for each engagement class.



The learned coefficients offered insight into feature importance across engagement classes. For instance, higher skin conductance increased the likelihood of a "Low Engagement" prediction while lowering "Moderate Engagement." However, as a linear model, Logistic Regression applies fixed weights and cannot capture feature interactions or nonlinear patterns likely present in biometric data.

To evaluate performance, I calculated macro-averaged metrics to account for class imbalance– these metrics are used across the report. The model achieved 64% accuracy, 0.63 precision, 0.62 recall, 0.61 F1-score, and a macro ROC-AUC of 0.86– indicating reasonable ranking ability despite frequent misclassification of high and moderate engagement levels. While interpretable and better than chance, Logistic Regression ultimately failed to capture the complexity of engagement patterns, making it a useful but limited baseline for comparison to more flexible models later in the report.

3.2. Support Vector Machine

To address the limitations of linear models, I implemented Support Vector Machines (SVM) using scikit-learn's `SVC`. The linear kernel achieved 68% accuracy, slightly better than Logistic Regression, but was still constrained by linear boundaries. To improve flexibility, I applied an Radial Basis Function (RBF) kernel, which enables nonlinear separation by implicitly mapping features into higher-dimensional space.

For the RBF-SVM, I tuned the `C` (regularization strength) and `gamma` (kernel width) hyperparameters using a grid search with 5-fold cross-validation.

Figure 3: Python implementation of RBF-kernel Support Vector Machine

```python
# Define parameter grid for SVM
param_grid = {
    'C': [0.1, 1, 10, 100],        # regularization strength
    'gamma': ['scale', 0.01, 0.1, 1],  # kernel coefficient
    'kernel': ['rbf']              # Radial Basis Function kernel
}

# Initialize SVM model
svm = SVC(probability=True)

# Set up grid search with 5-fold cross-validation
grid = GridSearchCV(svm, param_grid, cv=5, scoring='accuracy', verbose=1)
grid.fit(X_train, y_train)

# Best model and test set accuracy
print("Best Parameters:", grid.best_params_)
best_svm = grid.best_estimator_

y_pred = best_svm.predict(X_test)
print("SVM Test Accuracy:", accuracy_score(y_test, y_pred))
```

To quantitatively evaluate the improvement offered by the RBF kernel, I compared the performance of both models across multiple metrics.

Table 1: *Metrics comparing Linear vs RBF SVMs*

Metric	Linear SVM	RBF SVM
Accuracy	68.0%	88.5%
Precision (Macro)	0.67	0.87
Recall (Macro)	0.66	0.86
F1 Score (Macro)	0.66	0.86
ROC-AUC (Macro OvR)	0.88	0.96

The linear SVM achieved 68% accuracy, while the RBF-SVM improved this to 88.5%. Macro-averaged precision, recall, F1-score, and ROC-AUC all saw ~20-point gains, indicating better generalization across classes and stronger predictive confidence. These improvements support the use of a nonlinear kernel for this task.

To better understand how the model performed in feature space, I visualized its predictions using Principal Component Analysis (PCA) to reduce the 5-dimensional input data to 2 components.

Figure 4: PCA-reduced visualization of RBF-SVM predictions.

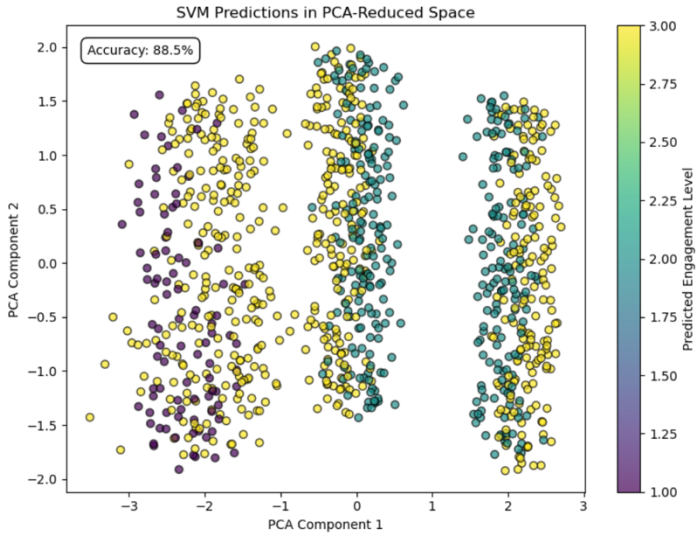


Figure 4 shows that PCA projection reveals clear clustering by predicted engagement level, suggesting the RBF-SVM successfully captured nonlinear structure in the data. This supports the performance gap observed between the linear and RBF models, emphasizing the value of flexible decision boundaries for complex physiological signals.

3.3. Random Forest

To evaluate a non-parametric, ensemble-based approach, I implemented a Random Forest Classifier using the `RandomForestClassifier` module from `sklearn.ensemble`. Random Forest constructs multiple decision trees using random subsets of the data and features, then aggregates their predictions by majority vote. This method is particularly useful for capturing complex nonlinear relationships, handling mixed data types, and being robust to overfitting due to its averaging nature.

I tuned hyperparameters—specifically the number of trees (`n_estimators`), tree depth (`max_depth`), and the minimum number of samples required to split (`min_samples_split`)—via 5-fold `GridSearchCV`. The final tuned model selected parameters that balanced tree depth and ensemble size, leading to improved generalization on the test set.

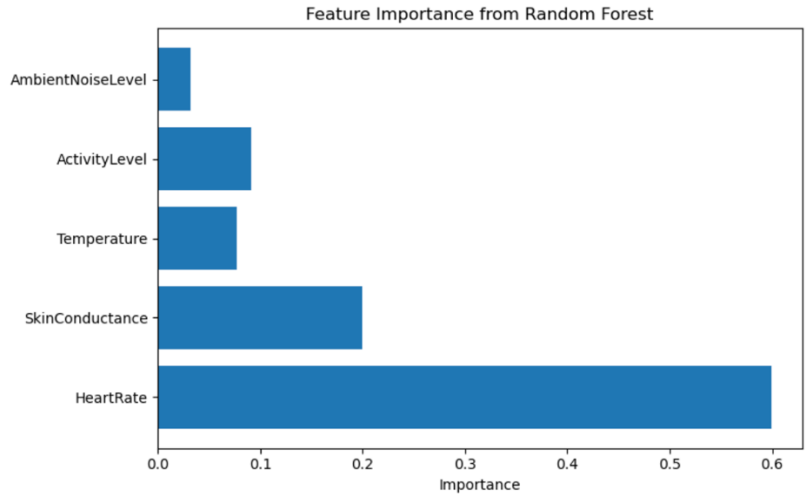
Figure 5: Random Forest Implementation Code Block.

```
# Define hyperparameter grid
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5]
}

# Run GridSearchCV to find best parameters
grid_rf = GridSearchCV(
    RandomForestClassifier(random_state=42),
    param_grid,
    cv=5,
    scoring='accuracy'
)
grid_rf.fit(X_train, y_train)
best_rf = grid_rf.best_estimator_

# Predict using best RF model
y_pred_rf = best_rf.predict(X_test)
```

Across evaluation metrics, Random Forest showed strong and balanced performance. It achieved a test accuracy of 88.5%, with macro-averaged precision**,** recall, and F1-score each exceeding 0.85, indicating that the model performed consistently well across all engagement classes. The ROC-AUC score of 0.97 further confirmed the model's ability to rank predictions confidently and correctly.

Unlike linear models, Random Forest does not produce easily interpretable coefficients, but it does provide feature importance scores.

Figure 5: Random Forest Feature Importance Plot



These scores indicated that Heart Rate and Skin Conductance were the most influential predictors, aligning with physiological expectations around arousal and attentiveness. Compared to the RBF SVM, which achieved the same accuracy, Random Forest provided the added benefit of feature interpretability, making it especially valuable for understanding the physiological drivers of engagement.

3.4. Extreme Gradient Boosting

To further explore ensemble methods, I implemented the XGBoost classifier using the `xgboost` Python library. XGBoost is a gradient boosting framework optimized for speed and performance, especially suited to structured data. I began with the default model and later performed hyperparameter tuning using `GridSearchCV`, optimizing parameters like `n_estimators`, `max_depth`, `learning_rate`, and `subsample`.

Figure 6: XGBoost Hyperparameter Tuning Code Block.

```
# Define parameter grid for tuning
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.05, 0.1],
    'subsample': [0.8, 1.0]
}

grid_xgb = GridSearchCV(
    estimator=XGBClassifier(eval_metric='mlogloss', random_state=42),
    param_grid=param_grid,
    cv=5,
    scoring='accuracy'
)

grid_xgb.fit(X_train, y_train_adj)

# Use the best model
best_xgb = grid_xgb.best_estimator_
y_pred_xgb = best_xgb.predict(X_test)
y_pred_xgb_final = y_pred_xgb + 1  # Shift back if needed
```

Interestingly, while I expected tuning to yield better performance, the test accuracy dropped slightly from 90% to 88%, with declines in precision, recall, and F1 score as well.

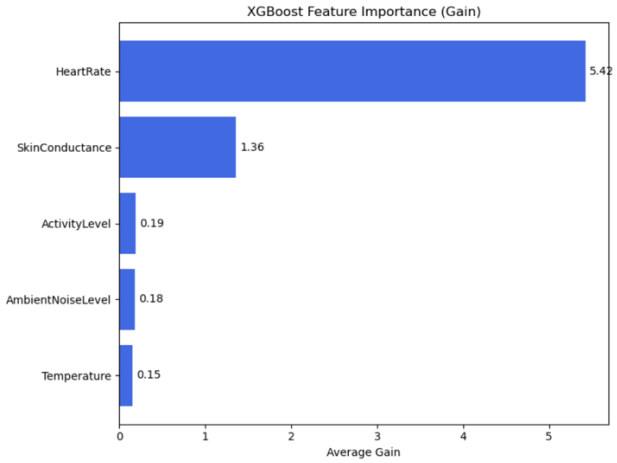
Table 2: Metrics comparing XGBoost Model Before and After Hyperparameter Tuning

Metric	Before Tuning	After Tuning
Accuracy	90.0%	88.0%
Precision (macro)	0.843	0.807
Recall (macro)	0.822	0.780
F1 Score (macro)	0.831	0.790
ROC-AUC (macro, OvR)	0.976	0.975

The slight drop in all metrics after hyperparameter tuning may reflect overfitting on cross-validation folds or a trade-off where reduced model variance came at the cost of accuracy. Since the untuned model already performed well, the tuning process may have introduced constraints (e.g., lower `subsample` or `max_depth`) that limited generalization. Hence, in my conclusions, I will be referring to the default model.

To better understand the model's behavior, I also visualized the learned feature coefficients for each class.

Figure 7: Feature Importance Plot from Default XGBoost Model



This feature importance plot helped confirm prior findings from Random Forest—highlighting heart rate and skin conductance as the most influential predictors of engagement state.

Despite the minor drop in performance after tuning, XGBoost remained one of the most powerful and reliable models in this analysis, balancing predictive accuracy with interpretability through feature importance.

4. Conclusion

To compare model performance statistically, I conducted 1,000 rounds of bootstrapped resampling. The mean accuracies and 95% confidence intervals (CI) are shown below:

Table 3: Bootstrapped Performance Metrics Across Models

Model	Mean Accuracy	2.5% CI	97.5% CI
Logistic Regression	0.640	0.575	0.700
Random Forest	0.885	0.840	0.925
XGBoost	0.900	0.855	0.940
SVM (RBF)	0.885	0.840	0.925

The bootstrapped accuracy summary confirmed that the nonlinear models, XGBoost, SVM (RBF), and Random Forest, achieved high and consistent performance across resampled test sets, with narrow confidence intervals indicating model stability.

Pairwise p-values (two-sided) were calculated to assess whether differences between models were statistically significant:

Table 4: Pairwise Model Comparison p-values

Comparison	p-value
Logistic Regression vs RF	0.4670
Logistic Regression vs XGBoost	0.4830
Logistic Regression vs SVM	0.4830
Random Forest vs XGBoost	0.4020
Random Forest vs SVM	0.8990
XGBoost vs SVM	0.6440

Despite XGBoost achieving the highest mean accuracy (90.0%), the differences between XGBoost, Random Forest, and SVM were not statistically significant ($p > 0.4$ in all cases). Therefore, all three models performed comparably well under resampled evaluation. Logistic Regression, while interpretable, lagged behind in predictive power.

While the differences among XGBoost, Random Forest, and SVM were not statistically significant, the bootstrapped results suggest a practical ranking: XGBoost slightly outperformed others, with SVM and Random Forest close behind.

Although the overall results are promising, a key limitation lies in assuming physiological signals always reflect meaningful engagement– users might be focused on irrelevant stimuli, limiting real-world applicability.

Still, the project showed that multimodal biometric data can classify engagement levels with high accuracy. Random Forest and SVM balanced performance and interpretability well,

while XGBoost delivered the strongest results overall. This pipeline offers a promising foundation for attention-aware wearable systems.