Gesture Phase Segmentation using Machine Learning Techniques

Reynaldo Ezequiel Ismesh Cabezas del Castillo
rcabezas@uamv.edu.ni

Fabricio Meneses Campos
fmeneses@uamv.edu.ni

May 2025

Abstract

This study addresses the challenge of automatic gesture phase segmentation, which involves distinguishing between distinct phases of human gestures such as Rest, Preparation, Stroke, Hold, and Retraction. Using a dataset recorded with Microsoft Kinect, we explore the segmentation of gestures in the context of natural speech and behavior, where subtle motion transitions between phases are key. The dataset consists of seven videos from three subjects narrating comic strips, with each gesture phase annotated based on motion features including spatial position, velocity, and acceleration of key joints. Categorical variables, such as the subject and the story being read, were encoded via one-hot encoding to facilitate effective machine learning modeling.

The impact of outliers on model performance was examined by creating three distinct datasets: one where all records containing outlier features were removed, one where only records with at most one outlier feature were retained, and one where records with at most two outlier features were kept. Various machine learning algorithms were evaluated, including Support Vector Machines (SVM), Perceptron, Decision Trees, Random Forests, Extra Trees, Naive Bayes, Gradient Boosting, and XGBoost, with hyperparameter optimization performed for each model. The models were assessed using the macro F1-score, a suitable metric for the multiclass, imbalanced nature of the task. All performance metrics were averaged over 5-fold cross-validation to ensure robust evaluation.

Results indicate that both SVM and Perceptron outperformed other classifiers when trained with default hyperparameters. Specifically, SVM achieved an initial macro F1-score of 0.3494, which improved to 0.3864 after tuning, representing a 10.58% performance gain. Similarly, Perceptron demonstrated a significant improvement, with an initial score of 0.3289 rising to 0.3994, marking a 21.43% increase. Since the performance across the three dataset configurations showed no disruptive differences, the "all outliers removed" dataset was selected for final model tuning due to its cleanliness and convenience. These findings underscore the importance of outlier management, feature encoding, and model simplicity in achieving robust performance. While cross-validation provides a reliable estimate, future work should explore model evaluation using a fixed training/test split to assess generalization under real deployment conditions.

1 Introduction

Human communication extends beyond speech—gestures carry substantial communicative weight, particularly in natural conversation. Segmenting these gestures into distinct temporal phases (Rest, Preparation, Stroke, Hold, and Retraction) is crucial for both linguistic and computational analysis. The task is challenging due to ambiguous phase boundaries, gesture variability across individuals, and subtle motion transitions.

This study focuses on automatic gesture phase segmentation using a labeled dataset recorded via Microsoft Kinect. The dataset comprises seven videos of three individuals narrating comic strips, with an-

notations and feature vectors capturing spatial, velocity, and acceleration data from key joints.

Given the labor-intensive and subjective nature of manual segmentation, automating this task enhances scalability and consistency for broader gesture analysis applications—spanning human-computer interaction, behavior recognition, and multimodal communication research.

2 Problem Statement

Despite significant advancements in gesture recognition, the temporal segmentation of gesture phases remains a critical and unresolved challenge. Unlike static gesture classification, phase segmentation must distinguish between subtle transitions—such as from a resting position to preparatory movement—often based on noisy and overlapping spatiotemporal signals.

This problem is particularly relevant in applications where precise gesture boundaries are essential: sign language interpretation, assistive technologies for motor-impaired individuals, and real-time human-computer interaction. In these contexts, small variations in movement can convey different intentions or meanings, making segmentation accuracy a fundamental requirement.

Using a Kinect-captured dataset annotated with five gesture phases, this project addresses the task of automatically identifying the correct gesture phase at each video frame, leveraging machine learning models trained on motion features like position, velocity, and acceleration.

3 Objectives

The goal of this project is to build and evaluate a machine learning-based classifier for the task of gesture phase segmentation. Specifically, we aim to:

- Explore and compare the performance of several supervised learning models, including: Decision Trees, Random Forests, Extra Trees, Support Vector Machines (SVM), Naive Bayes, Perceptron, Gradient Boosting, and XGBoost.
- Perform hyperparameter optimization for each algorithm to ensure fair and robust comparison.
- Use the F1-score Macro as the primary metric to evaluate classification performance due to the class imbalance and the multiclass nature of the problem.
- Identify the most effective model for accurately segmenting gesture phases based on spatio-temporal features.

4 Data Description

The dataset used in this project consists of motion capture recordings obtained via Microsoft Kinect sensors, tracking the 3D coordinates of upper-body joints as individuals perform hand gestures. Each gesture sequence is annotated frame by frame with one of five phases: Rest, Preparation, Stroke, Hold, or Retraction.

4.1 Raw Data

The raw dataset contains 20 attributes per frame:

- **Joint Positions (x, y, z):** Left hand (lhx, lhy, lhz), right hand (rhx, rhy, rhz), head (hx, hy, hz), spine (sx, sy, sz), left wrist (lwx, lwy, lwz), right wrist (rwx, rwy, rwz).
- **Timestamp:** A time marker for each frame.
- **Phase:** Categorical label indicating the gesture phase: Rest, Preparation, Stroke, Hold, or Retraction.

4.2 Processed Data

The processed dataset extends the raw attributes by computing motion-based features:

- **Vectorial Velocity and Acceleration (x, y, z):** For both hands and both wrists. These reflect the directional motion dynamics.

- **Scalar Velocities:** Magnitude of motion for hands and wrists, useful for understanding intensity without direction.
- **Phase Labels:** Labeled under the column `Phase` using a single-letter code: D (Descanso), P (Preparación), S (Stroke), H (Hold), R (Retracción).

The inclusion of kinematic variables like velocity and acceleration enables the modeling of temporal dynamics crucial for gesture phase segmentation. During preprocessing, only the `Phase` label was retained for consistency across the dataset.

5 Exploratory Data Analysis

The exploratory phase began with a structural review of the dataset, including verification of data types and consistency of numerical values. Additionally, two new categorical features were engineered from the filenames: **Subject** (a, b, c) and **Story** (1, 2, 3), derived from identifiers such as "a1", "b3", etc., which represent the performer and the narrative being enacted.

5.1 Descriptive Statistics

Summary statistics—such as mean, median, standard deviation, skewness, and kurtosis—were computed for all numerical features to characterize their distributions. This aligns with the principles of exploratory data analysis introduced by Tukey, which emphasize the use of descriptive metrics and visual summaries to uncover patterns and anomalies in data (Tukey, 1977).

5.2 Outlier Detection

Two well-established statistical approaches were evaluated for detecting outliers in the dataset:

- **Z-score method:** The z-score was computed for each numerical feature, and records with absolute z-scores greater than 3 were flagged as potential outliers. This approach relies on the assumption of normality, where the empirical rule suggests that approximately 99.7% of data should lie within three standard deviations from the mean (Shailesh, 2022).
- **Interquartile Range (IQR) method:** This method flagged data points outside the range of $[Q1 - 1.5 \times IQR, Q3 + 1.5 \times IQR]$, where $Q1$ and $Q3$ represent the first and third quartiles, respectively (Shailesh, 2022).

A comparative analysis using a plot (Figure 1) revealed that the IQR method would exclude a significantly larger portion of the data. Given the objective of preserving potentially meaningful gesture patterns while still mitigating the influence of extreme values, the z-score method was selected for outlier removal.

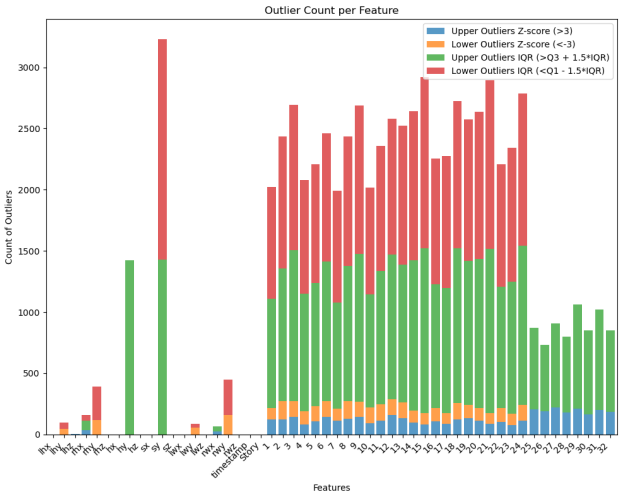

Figure 1: Comparison of outliers detected by Z-score vs. IQR methods

5.3 Distribution Analysis

Histograms were plotted for all numerical features to assess distribution shape. Particular attention was given to multimodal features, which may indicate complex gesture transitions or temporal phase overlaps. These features were earmarked for deeper analysis and potential use in feature engineering.

A representative grid of histograms is shown in Figure 2, highlighting variables visually identified as exhibiting multimodal distributions. The presence of multiple peaks suggests potential heterogeneity in the underlying signal, possibly corresponding to discrete phases or gesture types.

The following variables were identified as multimodal:

```
sy, sz, timestamp, lwz, rwx, rhx, rwy, rhy,
      sx, hx, hz, lhx, lhy, lwx, lhz
```

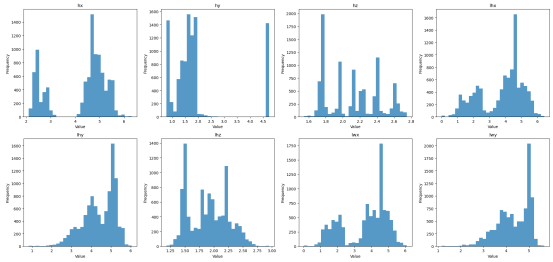

Figure 2: Histogram grid highlighting variables with multimodal distributions.

5.4 Outlier Removal

To identify and handle outliers in the dataset, we used the Z-score method instead of the Interquartile Range (IQR) method. The decision was based on a visual comparison of both techniques, which revealed that the IQR method would remove a significantly larger portion of the data, potentially compromising the model's learning capacity. The Z-score method considers a feature as an outlier when its absolute Z-score exceeds 3.

We quantified how many records would be removed based on the number of outlier features present in each observation. The table below shows how the dataset is distributed depending on how many features per record exceed the Z-score threshold:

Table 1: Record Count by Number of Outlier Features

# Outlier Features	Records	Cumulative	%
1	697	697	7.04
2	664	1361	13.75
3	285	1646	16.62
4	219	1865	18.84
5	116	1981	20.01
6–9	301	2282	23.05
10–15	78	2360	23.84
16–20	8	2368	23.92
21–25	0	2368	23.92
26–30	0	2368	23.92
31–35	0	2368	23.92
36–40	0	2368	23.92
41–45	0	2368	23.92
46–50	0	2368	23.92

We tested three outlier removal strategies:

- Removing all records that contain any outlier features.
- Keeping records with only one outlier feature.
- Keeping records with up to two outlier features.

6 Feature Selection

After applying the three outlier removal strategies—(i) removing all records containing any outlier features, (ii) keeping records with only one outlier feature, and (iii) keeping records with up to two outlier features—we proceeded to refine the feature space of each resulting dataset through correlation-based feature selection.

To begin, we generated Pearson correlation matrices for each version of the dataset in order to visualize and quantify the linear relationships between numerical features. Pairs of features exhibiting Pearson correlation coefficients above 0.90 were flagged as highly collinear. In the presence of such strong correlations, one feature can often be linearly predicted from another, which not only inflates variance in model coefficients but also introduces redundancy, reducing the distinctiveness of features and potentially degrading generalization performance (Hall, 1999).

The following variables exhibited Pearson correlation scores greater than 0.90, indicating a high degree of linear relationship between them:

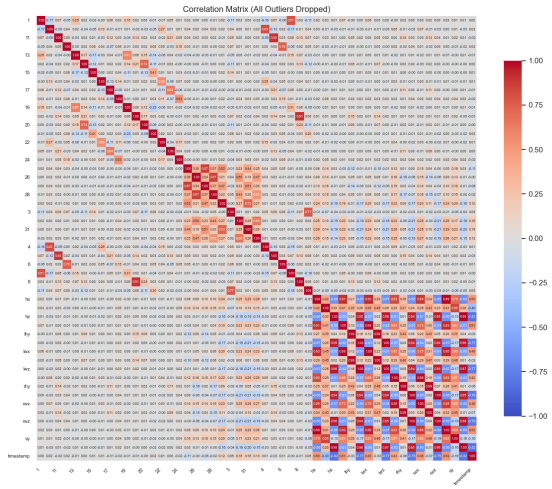

Figure 3: Pearson Correlation Matrix for Dataset with Outliers Removed

- `11 (Vectorial velocity of right wrist, y coordinate)` and `5 (Vectorial velocity of right hand, y coordinate)`
- `2 (Vectorial velocity of left hand, y coordinate)` and `8 (Vectorial velocity of left wrist, y coordinate)`
- `hx (Position of head, x coordinate)` and `lwx (Position of left wrist, x coordinate)`

These strong correlations can be explained as follows:

- **`11` and `5 (Right wrist and right hand velocity in y-axis)`:** The movement of both the wrist and hand along the y-axis is inherently connected, as the wrist is typically positioned at the end of the hand. Any motion of the hand in the y-axis will likely influence the wrist's movement in the same direction, leading to a high correlation between the two.
- **`2` and `8 (Left hand and left wrist velocity in y-axis)`:** Similarly, the left hand and wrist velocities are highly related because they both belong to the same limb. Movements of the hand will cause corresponding changes in wrist velocity, especially in the same direction.
- **`hx` and `lwx (Head position and left wrist position in x-axis)`:** The left wrist's position in the x-axis and the head position may be correlated because of the relative positioning of the body. As the person moves their left hand or arm, the head often shifts in tandem with the body's motion. The x-coordinate of both the head and the left wrist can thus exhibit a significant relationship during gestures.

To address these high correlations, we implemented a systematic pruning strategy: for each pair of highly correlated features, we calculated the average Pearson correlation of each feature with all other remaining features. The feature with the higher average correlation to the rest was removed, following a redundancy-reduction heuristic intended to preserve the more informative and less redundant feature of each highly correlated pair (Hall, 1999).

Finally, any remaining records with missing values were removed. The outcome of this process was three distinct, cleaned, and reduced datasets, each prepared for downstream model development with minimized multicollinearity and improved feature relevance.

7 Feature Engineering

The goal of the feature engineering phase was to enrich the dataset by creating new variables that could capture complex interactions between existing numerical features. This was accomplished by generating **interaction features** through basic arithmetic operations—addition, subtraction, and multiplication—on all possible pairs of numerical variables (Kuhn & Johnson, 2022).

Formally, for two numerical features A and B, new variables such as $A + B$, $A - B$, and $A \times B$ were introduced. The intuition behind this strategy lies in uncovering non-obvious patterns or relationships

between variables that may not be evident when analyzed in isolation.

Categorical features, on the other hand, were excluded from this step, as applying direct arithmetic operations to them would not yield meaningful or interpretable results.

While this approach resulted in a significantly expanded feature space, it is important to note that correlation analysis was not conducted among the newly generated interaction features. This remains a limitation, as some of these engineered variables could be highly correlated, leading to multicollinearity issues that might negatively impact model interpretability or stability (Kuhn & Johnson, 2022).

Despite this, the diversity introduced by combining heterogeneous features reduces the risk of systematic redundancy. Many resulting variables are expected to offer distinct information, even if only a subset will ultimately prove useful for prediction. Preliminary modeling suggested that these interaction features neither significantly harmed model performance nor introduced instability, and thus their inclusion was deemed beneficial.

In future iterations, applying correlation-based pruning or regularization techniques (e.g., Lasso) could further refine the engineered feature set by identifying and eliminating redundant variables.

8 Final Data Processing

The final data processing phase focused on ensuring the dataset was clean, complete, and ready for the machine learning models. This involved handling missing values, encoding categorical variables, and standardizing numerical variables to make the data suitable for model training and prediction.

8.1 Handling Missing Data

A critical aspect of preparing the data was ensuring that no records with missing or blank values remained in the dataset. Most machine learning algorithms require a complete dataset to function properly, as missing values can lead to errors or suboptimal model performance. Therefore, any record with a missing feature, across any of the pictures (treated as vectors in this context), was removed. This step ensured that all input data were valid, and no missing values would negatively affect the performance of the subsequent models.

8.2 Treatment of Numerical Variables

Once the missing values were handled, the numerical variables underwent standardization. Standardization is a key preprocessing step, particularly when the model relies on distance-based measures (e.g., k-nearest neighbors, support vector machines) or gradient-based optimization (e.g., neural networks). The numerical features were standardized using the Z-score, which transforms each feature to have a mean of 0 and a standard deviation of 1. This ensures that all features are on the same scale, reducing bias in models that might otherwise be sensitive to the magnitude of certain variables.

Additionally, numerical variables with extreme values or outliers were addressed earlier in the preprocessing pipeline, ensuring that these outliers do not significantly skew model results.

8.3 Treatment of Categorical Variables

8.3.1 Subject Encoding

The subject variable, which initially represented different subjects in the dataset (subject A, B, and C), was encoded using one-hot encoding. This method was chosen as the appropriate approach since the variable is nominal and does not carry an inherent order or ranking. One-hot encoding creates a new binary feature for each category (i.e., subject A, subject B, and subject C). This way, the machine learning models can interpret the variable without assuming any ordinal relationship between the subjects, allowing for more flexibility in model interpretation and performance.

8.3.2 Story Encoding

The story variable, which was initially represented as a numerical value (e.g., story 1, story 2, story 3),

was also categorical in nature, as it simply identifies the story without implying any quantitative relationship. This variable was treated similarly to the subject variable and was encoded using one-hot encoding as well. By creating binary features corresponding to each unique story (i.e., story 1, story 2, and story 3), the model could effectively handle this categorical information without assuming any ordinal or numerical significance in the original labels.

8.3.3 Phase Encoding (Target Variable)

The target variable, *phase*, originally represented a categorical variable with the following values:

- D: Rest position (from Portuguese "descanso")
- P: Preparation
- S: Stroke
- H: Hold
- R: Retraction

Each of these categories was initially assigned a numerical value from 1 to 5. However, it was later realized that some machine learning algorithms, especially those used for regression or classification, work better when the target variable starts from 0 rather than 1. Consequently, the target variable was shifted so that the categories would begin from 0. For instance, D (rest position) was changed from 1 to 0, P (preparation) from 2 to 1, and so on. This change ensured better compatibility with algorithms that expect a zero-indexed target.

8.4 Summary

In this final data processing stage, a careful and systematic approach was taken to clean and prepare the data for model training. Missing values were removed to ensure that no incomplete records were included in the model. Categorical variables such as *subject* and *story* were one-hot encoded to allow the machine learning models to interpret them effectively, while the *phase* variable was adjusted to ensure it aligned with the requirements of specific algorithms. Finally, numerical features were standardized using the Z-score to ensure uniformity in the data and to support models that rely on distance or gradient-based methods. These steps were essential in preparing the dataset for the next phase of analysis and modeling.

9 Baseline Model Performance Analysis

In this section, we evaluate the performance of eight different machine learning models using a 5-fold cross-validation strategy. The models were evaluated on three different datasets, each corresponding to different outlier handling strategies. The importance of evaluating different strategies for outlier handling lies in their potential to impact the generalizability and robustness of the models. Removing or keeping outliers can influence model stability and performance (Shailesh, 2022). The datasets correspond to the following outlier handling strategies:

- All outliers removed
- Two or more outliers kept
- Three or more outliers kept

9.1 Cross-Validation Process

A 5-fold cross-validation strategy was employed to assess the generalization capability of each model. This method divides the dataset into five subsets or "folds." For each iteration, one subset serves as the test set, while the remaining four subsets are used for training the model. This process is repeated five times, and the model's performance is averaged over all the folds. The final evaluation metric for each model is the mean F1 score across all five folds (Brownlee, 2014).

The F1 score, which is the harmonic mean of precision and recall, is a particularly valuable metric when working with imbalanced datasets. It provides a balanced measure by accounting for both false positives and false negatives, making it more informative than accuracy in many real-world scenarios, es-

pecially when the costs of false positives and false negatives are not equal (Hall, 1999).

The models evaluated in this analysis were:

- Decision Tree
- Random Forest
- Gradient Boosting
- XGBoost
- Support Vector Machine (SVM)
- Extra Trees
- Naive Bayes
- Perceptron

These models were selected to provide a diverse set of algorithms, ranging from simple tree-based methods (Decision Tree, Random Forest) to more complex ensemble and boosting techniques (Gradient Boosting, XGBoost). This variation allows us to explore how different models handle the feature space after outlier treatment (Bergstra & Bengio, 2012).

9.2 Model Performance Metrics

For classification performance evaluation across imbalanced classes, we used the **macro-averaged F1 score**, which assigns equal importance to all classes, regardless of their frequency in the dataset. This is particularly important when dealing with operational scenarios where minority classes may have a disproportionate impact on decisions, such as fraud detection or medical diagnosis (Raschka, n.d.). The macro-averaged F1 score helps to avoid biasing the model toward the majority class and ensures that the performance on the minority class is adequately measured (Hall, 1999).

All models were evaluated using 5-fold cross-validation on three differently filtered datasets, each reflecting a specific level of outlier removal. These levels include: (1) all outliers removed, (2) only those with two or more outlier features removed, and (3) only those with three or more outlier features removed. By assessing model performance across these different outlier-handling scenarios, we can better understand how each model reacts to variations in data quality and its sensitivity to outliers (Shailesh, 2022).

9.3 Results and Visualizations

Below is the comparative performance of eight models across the three datasets. The table displays the average macro F1 scores, while the bar chart provides a visual comparison of model performance sensitivity to outlier handling.

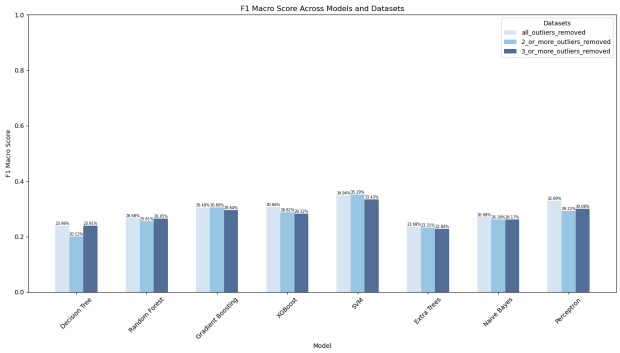

Figure 4: Average F1 Macro Scores Across Outlier Filtering Levels

9.4 F1 Scores Table

Model	All Feature Outliers Dropped	2 or more Feature Outliers Dropped	3 or more Outliers Dropped
Decision Tree	0.239784439	0.201225824	0.239087616
Random Forest	0.266836283	0.256099366	0.264501940
Gradient Boosting	0.304926452	0.305958834	0.296397783
XGBoost	0.306587672	0.288205475	0.283161355
Support Vector Machine (SVM)	0.349420674	0.352879967	0.334328590
Extra Trees	0.236773581	0.233115095	0.228439621
Naive Bayes	0.269789277	0.261842300	0.261748665
Perceptron	0.328949014	0.292194605	0.300843622

Table 2: Average F1 Macro Scores of Models across Datasets

9.5 Model Selection Rationale

As shown in Table 2, Support Vector Machine (SVM) and Perceptron were the models with the

highest macro-averaged F1 scores across the different outlier-handling strategies. SVM reached its best performance (0.3528) on the dataset where only records with a single outlier feature were retained (i.e., records with two or more outlier features were dropped), while the Perceptron performed best (0.3289) when all records with any outlier feature were entirely removed.

Although other models such as XGBoost and Gradient Boosting showed competitive performance, they were ultimately excluded from further tuning in favor of computational efficiency and to keep the focus on two distinctly different learning algorithms that had already demonstrated promising results.

9.6 Next Steps: Focused Model Tuning

For consistency and practical reasons, both SVM and Perceptron were fine-tuned using the same dataset: the one in which all records with any number of outlier features were dropped ("All Feature Outliers Dropped"). This choice simplified the workflow while also offering a cleaner and less noisy dataset to train on — making it easier to evaluate the impact of hyperparameter tuning. The assumption here is that a cleaner dataset increases the likelihood of observing more stable performance improvements, especially during cross-validation.

10 Hyperparameter Tuning for Selected Models

In this section, we present the hyperparameter tuning results for the two models selected based on their outstanding performance in initial benchmarking: **Support Vector Machine (SVM)** and **Perceptron**. These models showed the highest macro-averaged F1 scores, and were therefore chosen for further tuning to optimize their classification performance.

All hyperparameter tuning experiments were performed using the version of the dataset where all records containing outliers in any feature were removed. This ensured consistency, improved stability during training, and simplified comparisons across tuning methods.

10.1 Evaluation Strategy and Search Methods

To identify the optimal hyperparameters for each model, we used three different search algorithms, each with its own advantages:

- **Grid Search:** Exhaustively tests all possible combinations of the specified hyperparameter values. It is the most comprehensive method but computationally expensive.
- **Random Search:** Randomly samples a fixed number of hyperparameter combinations. This method is more efficient than Grid Search, especially for large hyperparameter spaces.
- **Bayesian Optimization:** A probabilistic, model-based optimization method that uses past evaluations to decide the next set of hyperparameters to try. It typically converges to optimal values faster than both Grid and Random Search.

All models were evaluated using 5-fold cross-validation and the macro-averaged F1 score as the primary metric. In order to keep the runtime reasonable, the number of iterations for both Random Search and Bayesian Optimization was set to 60% of the total number of valid hyperparameter combinations evaluated during Grid Search.

10.2 Hyperparameter Tuning for Support Vector Machine (SVM)

The SVM model was tuned using three kernel types: `linear`, `rbf/sigmoid`, and `poly`. Each kernel required a specific set of compatible hyperparameters. The dataset used for this tuning excluded all records containing outliers in any feature, a choice made to ensure cleaner decision boundaries and more reliable performance evaluation.

The hyperparameters explored for SVM are shown below:

Hyperparameter	Values Tested
Kernel	`linear`, `rbf`, `sigmoid`, `poly`
C (Regularization)	0.001, 0.01, 0.1, 1, 10, 100, 1000
Gamma	`scale`, `auto`, 0.001, 0.01, 0.1, 1
Degree (only for poly)	2, 3, 4, 5

Table 3: Hyperparameters tested for SVM model

To ensure meaningful results, incompatible combinations of hyperparameters were avoided. For instance:

- `gamma` was only used with `rbf`, `sigmoid`, and `poly` kernels.
- `degree` was only used with the `poly` kernel.

The number of iterations for both **Random Search** and **Bayesian Optimization** was set to 60% of the total number of valid hyperparameter combinations evaluated during Grid Search.

The table below summarizes the performance and computational cost of each search method used during tuning:

Search Method	Time (seconds)	Macro F1 Score
Grid Search	4480.03	0.3864
Random Search	3950.43	0.3609
Bayes Search	14009.14	0.3779

Table 4: SVM Hyperparameter Tuning Results

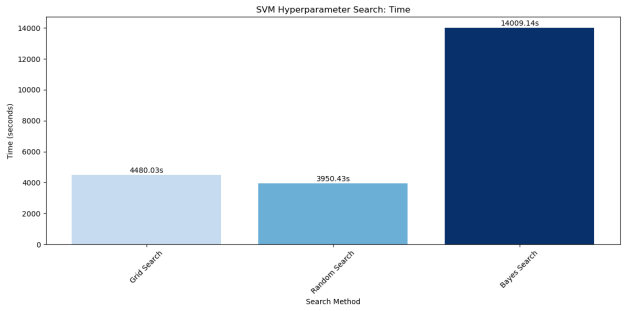

Figure 5: Search Time for SVM Tuning (Grid, Random, Bayesian)

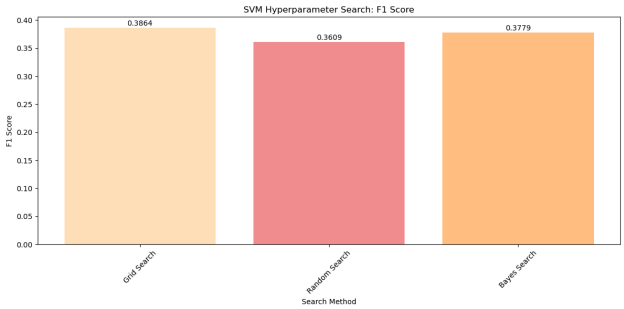

Figure 6: Macro F1 Score Achieved by Each Search Method for SVM

10.3 Hyperparameter Tuning for Perceptron

In this subsection, we explore the hyperparameter tuning results for the Perceptron model, leveraging three different search methods: Grid Search, Random Search, and Bayesian Optimization. The dataset used for the Perceptron tuning excluded all records with outliers across all features, ensuring cleaner data input for model training and evaluation.

Dataset Preparation: The dataset was loaded, keeping the features and target variable separate. A necessary preprocessing step involved adjusting the target label down by one.

10.3.1 Hyperparameter Exploration

The tuning involved defining and exploring an expanded hyperparameter space which includes penalty type, regularization parameters, maximum number of iterations, and convergence tolerance. The macro-averaged F1 score was selected as the evaluation metric.

Hyperparameter	Values Evaluated
Penalty	`l1`, `l2`, `elasticnet`
Alpha	0.00001, 0.0001, 0.001, 0.01, 0.1
Max Iterations	500, 1000, 1500, 2000
Tolerance	1e-4, 1e-3, 1e-2

Table 5: Hyperparameters evaluated for Perceptron model

10.3.2 Search Methods and Results

Three distinct search strategies were employed to identify optimal hyperparameter configurations:

- **Grid Search:** A thorough examination of all possible parameter combinations was performed, taking approximately 110.09 seconds.
- **Random Search:** Randomized sampling of hyperparameter space yielded results in a mere 14.83 seconds.
- **Bayesian Search:** Bayesian Optimization converged as effectively as Grid Search but in 30.09 seconds.

Search Method	Time (seconds)	Macro F1 Score
Grid Search	110.09	0.3994
Random Search	14.83	0.3680
Bayesian Search	30.09	0.3994

Table 6: Perceptron Hyperparameter Tuning Results

10.3.3 Outcome Visualization

The results were visualized using bar graphs to compare the efficiency (time) and effectiveness (F1 Score) of each search method for hyperparameter tuning in the Perceptron model.

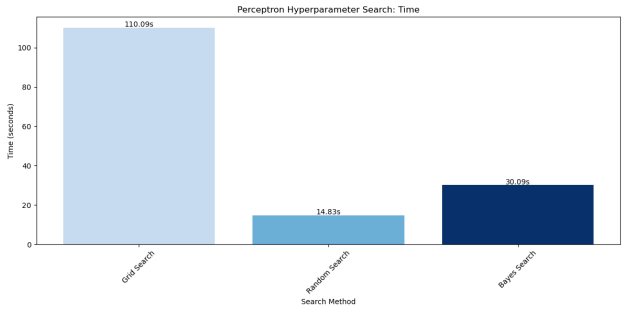

Figure 7: Search Time for Perceptron Tuning (Grid, Random, Bayesian)

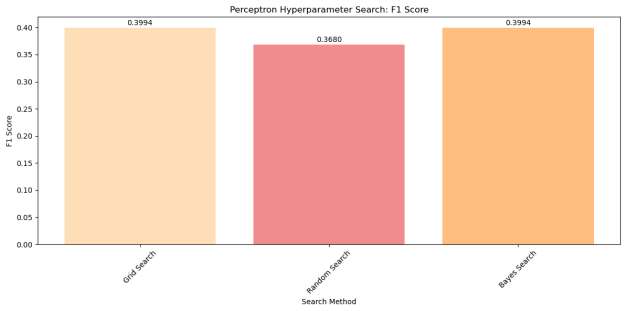

Figure 8: Macro F1 Score Achieved by Each Search Method for Perceptron

Both Grid Search and Bayesian Search achieved similar F1 scores, marking them as effective strategies for the Perceptron model despite the difference in time required. The visualizations further emphasize Bayesian Optimization as a favorable approach when balancing efficiency and performance.

11 Comparative Analysis: SVM vs. Perceptron with All Outliers Dropped

The dataset with all outliers removed provided an optimal environment for model evaluation and optimization. This section presents a comparative analysis between Support Vector Machine (SVM) and Perceptron models, with a focus on their performance under rigorous tuning conditions.

11.1 Performance Evaluation with "All Outliers Dropped"

Both models underwent hyperparameter tuning using the cleanest dataset configuration. The results were as follows:

- **Support Vector Machine (SVM):** The initial macro F1 score was 0.3494, which improved to 0.3864 following tuning, showing a significant performance improvement. The kernel-based

SVM, with its complexity arising from its ability to map data into higher-dimensional spaces, often leads to increased computational time and more complex optimization requirements, especially in high-dimensional spaces (Cortes & Vapnik, 1995).

- **Perceptron:** The initial score was 0.3289, with tuning improving the F1 score to 0.3994, a noticeable gain. Perceptrons, being linear classifiers, are computationally simpler compared to kernelized models like SVM, which contributes to faster optimization and less computational overhead.

Model	Initial	Post-Tuning	Increase	% Increase
SVM	0.3494	0.3864	0.0370	10.58%
Perceptron	0.3289	0.3994	0.0705	21.43%

Table 7: Performance Improvements Upon Tuning: Numeric and Percentage Gains

11.2 Comparative Insights

SVM demonstrated robust performance, with a notable 10.58% improvement in the F1 score. However, the Perceptron exhibited a higher adaptability, with a 21.43% increase after tuning, underlining its potential to achieve significant performance gains through meticulous hyperparameter refinement.

- **SVM Stability:** The SVM model's 10.58% improvement reflects its solid baseline capabilities. However, due to its kernel complexity, it demands higher computational resources, particularly during hyperparameter search and optimization (Cortes & Vapnik, 1995).

- **Perceptron's Adaptability:** The 21.43% increase in Perceptron's performance highlights its relatively simpler structure, enabling faster optimization with significant gains in performance after tuning.

11.3 Comparison of Hyperparameter Search Times

The following section illustrates the differences in time consumption between the hyperparameter search methods for SVM and Perceptron models, shedding light on the computational demands associated with each model.

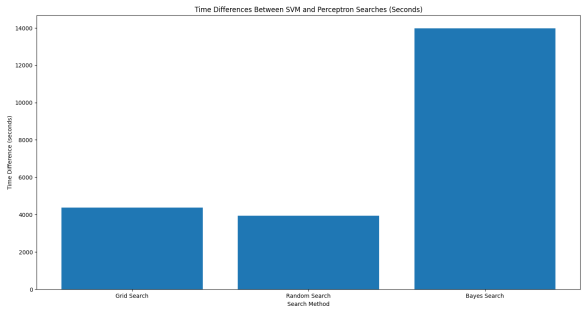

Figure 9: Time Differences Between SVM and Perceptron Searches (Seconds) (SVM - Perceptron)

Search Method	Numeric Difference (seconds)	Percentage Difference (%)
Grid Search	4369.94	3968.35%
Random Search	3935.60	26543.82%
Bayes Search	13979.05	46462.59%

Table 8: Numeric and Percentage Differences in Search Times Between SVM and Perceptron

Analysis of Search Time Differences The disparities in search times between SVM and Perceptron highlight the substantial computational overhead required by SVM during hyperparameter optimization. The SVM model's complexity, particularly with kernel methods, is a key factor behind the increased time consumption. For example, the Bayesian Search method required an additional 13,979.05 seconds (approximately 3.88 hours) for SVM compared to Perceptron, which represents a 46,462.59% increase in time consumption. This demonstrates the trade-off between the model's higher performance potential and the greater computational resources it demands, especially in kernelized models (Cortes & Vapnik,

1995).

12 Recommendations

To further explore potential performance, we recommend evaluating the SVM model using the dataset configuration that retains records with one outlier feature. This setup may allow SVM to better leverage its ability to handle slight noise, potentially improving both its adaptability and performance.

Additionally, we encourage exploring alternative evaluation metrics beyond the macro F1 score. Depending on the specific goals of the analysis, metrics such as the weighted F1 score, accuracy, AUC (Area Under the Curve), or lift could provide more comprehensive insights into the model's effectiveness, particularly for imbalanced datasets or cases where precision and recall trade-offs are crucial.

13 Limitations

Throughout the project, we encountered significant constraints linked to computation resources. Initial reliance on Google Colab Pro proved suboptimal, as GPU access was frequently inconsistent, leading to slower CPU-only runtimes and disconnections that disrupted workflow. Recognizing the need for enhanced computational power, we transitioned to AWS's `g5.xlarge` instance equipped with 4 vCPUs and upgraded to `g5.x2large` with 8 vCPUs, leveraging its capabilities for model training. However, this server setup required time to implement effectively, postponing immediate solutions. Despite attempts to utilize local machines, it became clear that specialized infrastructure was crucial for comprehensive, uninterrupted experimentation.

14 Conclusion

This study underscored the critical role of automatic gesture phase segmentation in human communication analysis, particularly in gesture recognition. Through extensive evaluation, hyperparameter tuning, and model comparison, the Perceptron emerged as the top-performing algorithm with optimal macro F1 scores post-tuning. The SVM remains promising, particularly with the potential for high performance in configurations retaining minimal outlier features.

Gesture phase segmentation has far-reaching implications, from sign language interpretation to enhanced human-computer interaction. By efficiently automating this process, our findings enable expansive, scalable analyses within linguistic research, assistive technologies, and communication fields. Future work should focus on refining model configurations further, maximizing outcomes with tailored datasets and continued exploration through enhanced computational resources.

References

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, *13*, 281–305.

Brownlee, J. (2014). *Data preparation for machine learning*. Machine Learning Mastery.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*, 273–297.

Hall, M. A. (1999). *Correlation-based feature selection for machine learning* (Doctoral dissertation, University of Waikato). Retrieved from `https://www.cs.waikato.ac.nz/~mhall/thesis.pdf`

Kuhn, M., & Johnson, K. (2022). *Feature engineering and selection: A practical approach for predictive models*. CRC Press.

Raschka, S. (n.d.). *How can the f1-score help with dealing with class imbalance?* `https://sebastianraschka.com/faq/docs/computing-the-f1-score.html`. (n.d.)

Shailesh. (2022). *Outliers detection using iqr, z-score, lof and dbscan.* `https://www.analyticsvidhya.com/blog/2022/10/outliers-detection-using-iqr-z-score-lof-and-dbscan/`. (Accessed: 2025-05-09)

Tukey, J. W. (1977). *Exploratory data analysis*. Addison-Wesley.