

## Model Optimization And Tuning Phase Report

Date	06 JULY 2024
Team ID	739909
Project Name	Unlocking Silent Signals: Decoding Body Language With Mediapipe
Maximum Marks	10 Marks

### Model Optimization and Tuning Phase:

The Model Optimization and Tuning Phase involves refining machine learning models for peak performance. It includes optimized model code, fine-tuning hyperparameters, comparing performance metrics, and justifying the final model selection for enhanced predictive accuracy and efficiency.

### Hyperparameter Tuning Documentation (6 Marks):

Model	Tuned Hyperparameters	Optimal Values
SVM	<pre># Create the SVM svm = SVC()  # Define the parameter grid for RandomizedSearchCV param_distributions = {     'C': [0.1, 1, 10, 100],     'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],     'gamma': ['scale', 'auto'] }</pre>	<pre># Get the best parameters and best score print("Best parameters from RandomizedSearchCV: ", random_search_svm.best_params_) print("Best score from RandomizedSearchCV: ", random_search_svm.best_score_)  # Assuming you have already defined and trained your best SVM model best_svm = random_search_svm.best_estimator_  # Make predictions on the training set y_tr = best_svm.predict(x_train) print("Training accuracy: ", accuracy_score(y_tr, y_train))  # Make predictions on the test set y_pred = best_svm.predict(x_test) print("Test accuracy: ", accuracy_score(y_pred, y_test))  Best parameters from RandomizedSearchCV: {'kernel': 'rbf', 'gamma': 'scale', 'C': 0.1} Best score from RandomizedSearchCV: 0.6166666666666667 Training accuracy: 0.6166666666666667 Test accuracy: 0.6</pre>
Logistic Regression	<pre># Create the LogisticRegression log_reg = LogisticRegression()  # Define the parameter grid for RandomizedSearchCV param_distributions = {     'C': [0.1, 1, 10, 100],     'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'] }</pre>	<pre># Get the best parameters and best score print("Best parameters from RandomizedSearchCV: ", random_search_log_reg.best_params_) print("Best score from RandomizedSearchCV: ", random_search_log_reg.best_score_)  # Assuming you have already defined and trained your best Logistic Regression model best_log_reg = random_search_log_reg.best_estimator_  # Make predictions on the training set y_tr = best_log_reg.predict(x_train) print("Training accuracy: ", accuracy_score(y_tr, y_train))  # Make predictions on the test set y_pred = best_log_reg.predict(x_test) print("Test accuracy: ", accuracy_score(y_pred, y_test))  Fitting 5 folds for each of 20 candidates, totalling 100 fits Best parameters from RandomizedSearchCV: {'solver': 'newton-cg', 'C': 0.1} Best score from RandomizedSearchCV: 0.6 Training accuracy: 1.0 Test accuracy: 0.6</pre>

Ridge Classifier	<pre># Create the RidgeClassifier ridge = RidgeClassifier()  # Define the parameter grid for RandomizedSearchCV param_distributions = {     'alpha': [0.1, 1, 10, 100],     'solver': ['auto', 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag', 'saga'] }</pre>	<pre># Get the best parameters and best score print("Best parameters from RandomizedSearchCV: ", random_search_ridge.best_params_) print("Best score from RandomizedSearchCV: ", random_search_ridge.best_score_)  # Assuming you have already defined and trained your best RidgeClassifier model best_ridge = random_search_ridge.best_estimator_  # Make predictions on the training set y_tr = best_ridge.predict(x_train) print("Training accuracy: ", accuracy_score(y_tr, y_train))  # Make predictions on the test set y_pred = best_ridge.predict(x_test) print("Test accuracy: ", accuracy_score(y_pred, y_test))  Best parameters from RandomizedSearchCV: {'solver': 'auto', 'alpha': 100} Best score from RandomizedSearchCV: 0.6 Training accuracy: 1.0 Test accuracy: 0.6</pre>
Gradient Boosting Classifier	<pre># Create the GradientBoostingClassifier gbc = GradientBoostingClassifier()  # Define a smaller parameter grid for RandomizedSearchCV param_distributions = {     'n_estimators': [50, 100],     'learning_rate': [0.01, 0.1],     'max_depth': [3, 4],     'min_samples_split': [2, 5],     'min_samples_leaf': [1, 2]</pre>	<pre># Get the best parameters and best score best_params_gbc = random_search_gbc.best_params_ best_score_gbc = random_search_gbc.best_score_ print("Best parameters from RandomizedSearchCV: ", best_params_gbc) print("Best score from RandomizedSearchCV: ", best_score_gbc)  # Assuming you have already defined and trained your best Gradient Boosting model best_gbc = random_search_gbc.best_estimator_  # Make predictions on the training set y_tr = best_gbc.predict(x_train) train_accuracy_gbc = accuracy_score(y_tr, y_train) print("Training accuracy: ", train_accuracy_gbc)  # Make predictions on the test set y_pred = best_gbc.predict(x_test) test_accuracy_gbc = accuracy_score(y_pred, y_test) print("Test accuracy: ", test_accuracy_gbc)  Best parameters from RandomizedSearchCV: {'n_estimators': 50, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_depth': 4, 'learning_rate': 0.1} Best score from RandomizedSearchCV: 0.807570411297888 Training accuracy: 0.94075 Test accuracy: 0.86</pre>
Random Forest Classifier	<pre># Create the RandomForestClassifier rf = RandomForestClassifier()  param_distributions = {     'n_estimators': [50, 100], # Reduced number of options     'criterion': ['gini', 'entropy'],     'max_features': ['auto', 'sqrt'],     'max_depth': [None, 10, 20], # Reduced number of options     'min_samples_split': [2, 5], # Reduced number of options     'min_samples_leaf': [1, 2] # Reduced number of options</pre>	<pre># Get the best parameters and best score best_params_rf = random_search.best_params_ best_score_rf = random_search.best_score_ print("Best parameters from RandomizedSearchCV: ", best_params_rf) print("Best score from RandomizedSearchCV: ", best_score_rf)  # Assuming you have already defined and trained your best random forest model best_rf = random_search.best_estimator_  # Make predictions on the training set y_tr = best_rf.predict(x_train) train_accuracy_rf = accuracy_score(y_tr, y_train) print("Training accuracy: ", train_accuracy_rf)  # Make predictions on the test set y_pred = best_rf.predict(x_test) test_accuracy_rf = accuracy_score(y_pred, y_test) print("Test accuracy: ", test_accuracy_rf)  Best parameters from RandomizedSearchCV: {'n_estimators': 50, 'min_samples_split': 5, 'min_samples_leaf': 2, 'max_features': 'auto', 'max_depth': 10, 'criterion': 'gini'} Best score from RandomizedSearchCV: 0.89253042657027 Training accuracy: 0.9825 Test accuracy: 0.9</pre>

## Performance Metrics Comparison Report (2 Marks):

Model	Optimized Metric
SVM	<pre>print("Classification report:\n", classification_report(y_test, y_pred))</pre> <pre>Classification report:               precision    recall  f1-score   support           0           0.83       0.92       0.87         93          1           0.93       0.83       0.88        107   accuracy          0.88  macro avg         0.88  weighted avg      0.88  Confusion Matrix: [[86  7]  [18 89]]</pre>

Logistic Regression	<pre>print("Classification report:\n", classification_report(y_test, y_pred))  Classification report:               precision    recall  f1-score   support           0       0.81      0.92      0.86         93          1       0.93      0.81      0.87        107   accuracy      0.86      200  macro avg      0.87      200 weighted avg      0.87      200  Confusion Matrix: [[86  7]  [20 87]]</pre>
Ridge Classifier	<pre>print("Classification report:\n", classification_report(y_test, y_pred))  Classification report:               precision    recall  f1-score   support           0       0.80      0.92      0.86         93          1       0.92      0.80      0.86        107   accuracy      0.86      200  macro avg      0.86      200 weighted avg      0.87      200  Confusion Matrix: [[86  7]  [21 86]]</pre>
Gradient Boosting Classifier	<pre>print("Classification report:\n", classification_report(y_test, y_pred))  Classification report:               precision    recall  f1-score   support           0       0.83      0.86      0.85         93          1       0.88      0.85      0.86        107   accuracy      0.85      200  macro avg      0.85      200 weighted avg      0.86      200  Confusion Matrix: [[80 13]  [16 91]]</pre>
Random Forest Classifier	<pre>print("Classification report:\n", classification_report(y_test, y_pred))  Classification report:               precision    recall  f1-score   support           0       0.82      0.95      0.88         93          1       0.95      0.82      0.88        107   accuracy      0.88      200  macro avg      0.88      200 weighted avg      0.89      200  Confusion Matrix: [[88  5]  [19 88]]</pre>

**Final Model Selection Justification (2 Marks):**

<b>Final Model</b>	<b>Reasoning</b>
Random Forest	The Random Forest model was selected for its robust performance, demonstrating high accuracy during hyperparameter tuning. Its ability to handle complex relationships, reduce overfitting through ensemble learning, and provide feature importance aligns with project objectives, justifying its selection as the final model.