

Model Development Phase Template

| | |
|---------------|---|
| Date | 21 July 2024 |
| Team ID | Team-740025 |
| Project Title | Unlocking Silent Signals :Decoding Body Language with Mediapipe |
| Maximum Marks | 10 Marks |

Initial Model Training Code, Model Validation and Evaluation Report

The initial model training code will be showcased in the future through a screenshot. The model validation and evaluation report will include a summary and training and validation performance metrics for multiple models, presented through respective screenshots.

Initial Model Training Code (5 marks):

Paste the screenshot of the model training code

Model Validation and Evaluation Report (5 marks):

| Model | Summary | Training and Validation Performance Metrics |
|-------|---------|---|
|-------|---------|---|

| | | |
|----------------|--|---|
| <p>Model 1</p> | <p>Gradient Boosting is the grouping of Gradient descent and Boosting. In gradient boosting, each new model minimizes the loss function from its predecessor using the Gradient Descent Method. This procedure continues until a more optimal estimate of the target variable has been achieved</p> | <h3>Train Machine Learning Classification Model</h3> <pre>[39]: from sklearn.pipeline import make_pipeline from sklearn.preprocessing import StandardScaler from sklearn.linear_model import LogisticRegression, RidgeClassifier from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier [40]: pipelines = { 'lr': make_pipeline(StandardScaler(), LogisticRegression()), 'rc': make_pipeline(StandardScaler(), RidgeClassifier()), 'rf': make_pipeline(StandardScaler(), RandomForestClassifier()), 'gb': make_pipeline(StandardScaler(), GradientBoostingClassifier()), } [41]: pipelines.keys() [42]: dict_keys(['lr', 'rc', 'rf', 'gb']) [43]: from sklearn.linear_model import LogisticRegression model = LogisticRegression(solver='lbfgs', max_iter=1000) model.fit(X, y) [44]: • LogisticRegression LogisticRegression(max_iter=1000) [45]: from sklearn.preprocessing import StandardScaler from sklearn.linear_model import LogisticRegression from sklearn.pipeline import make_pipeline</pre> |
| <p>Model 2</p> | <p>Logistic Regression is used for binary classification tasks, predicting outcomes like spam vs. non-spam or disease vs. no disease, and it provides probabilities of class membership. It also helps in understanding feature importance and serves as a strong baseline model due to its simplicity and interpretability.</p> | <pre>[43]: • Pipeline • StandardScaler • LogisticRegression [44]: from sklearn.linear_model import LogisticRegression model = LogisticRegression(solver='saga', max_iter=1000) model.fit(X, y) [45]: • LogisticRegression LogisticRegression(max_iter=1000, solver='saga') fit_models = {} for algo, pipeline in pipelines.items(): model = pipeline.fit(X_train, y_train) fit_models[algo] = model fit_models fit_models['rc'].predict(X_test) Evaluate and serialize model from sklearn.metrics import accuracy_score # Accuracy metrics import pickle for algo, model in fit_models.items():</pre> |
| <p>Model 3</p> | <p>The Random Forest Classifier is ideal because it combines the strength of multiple decision trees, offering high accuracy, robust performance against overfitting, and the ability to handle large and complex datasets, ensuring reliable predictions in varied shipping scenarios.</p> | <h3>Train Machine Learning Classification Model</h3> <pre>[39]: from sklearn.pipeline import make_pipeline from sklearn.preprocessing import StandardScaler from sklearn.linear_model import LogisticRegression, RidgeClassifier from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier [40]: pipelines = { 'lr': make_pipeline(StandardScaler(), LogisticRegression()), 'rc': make_pipeline(StandardScaler(), RidgeClassifier()), 'rf': make_pipeline(StandardScaler(), RandomForestClassifier()), 'gb': make_pipeline(StandardScaler(), GradientBoostingClassifier()), } [41]: pipelines.keys() [42]: dict_keys(['lr', 'rc', 'rf', 'gb']) [43]: from sklearn.linear_model import LogisticRegression model = LogisticRegression(solver='lbfgs', max_iter=1000) model.fit(X, y) [44]: • LogisticRegression LogisticRegression(max_iter=1000) [45]: from sklearn.preprocessing import StandardScaler from sklearn.linear_model import LogisticRegression from sklearn.pipeline import make_pipeline</pre> |

Model 4

The file "coords.csv" into a Pandas DataFrame. It then splits the data into training and testing sets using the train_test_split function from Scikit-learn. The df.head() and df.tail() functions are used to display the first and last few rows of the DataFrame, respectively.

* Training model using Scikit Learn

Read in collected data and process

```
[29]: import pandas as pd
      from sklearn.model_selection import train_test_split

[30]: df = pd.read_csv('coords.csv')

[31]: df.head()
```

| | class | x1 | y1 | x1 | v1 | x2 | y2 | x2 | v2 | x3 | ... | x499 | v499 | x500 | y500 | x500 | v500 | x501 |
|---|-------|----------|----------|-----------|---------|----------|----------|-----------|----------|----------|-----|-----------|------|----------|----------|----------|------|----------|
| 0 | Happy | 0.460887 | 0.595566 | -1.121984 | 0.99785 | 0.484884 | 0.513489 | -1.068172 | 0.999606 | 0.504262 | .. | -0.001009 | 0.0 | 0.532770 | 0.511281 | 0.031799 | 0.0 | 0.537714 |
| 1 | Happy | 0.461394 | 0.595729 | -1.218655 | 0.99784 | 0.483257 | 0.513369 | -1.173208 | 0.999604 | 0.501266 | .. | -0.000845 | 0.0 | 0.529603 | 0.512033 | 0.034787 | 0.0 | 0.534479 |
| 2 | Happy | 0.461530 | 0.595861 | -1.204079 | 0.99784 | 0.481376 | 0.513182 | -1.198528 | 0.999603 | 0.498597 | .. | -0.000800 | 0.0 | 0.528913 | 0.512119 | 0.034527 | 0.0 | 0.533444 |
| 3 | Happy | 0.463032 | 0.599466 | -1.253284 | 0.99765 | 0.481420 | 0.515375 | -1.181970 | 0.999566 | 0.498603 | .. | -0.001560 | 0.0 | 0.529767 | 0.511925 | 0.032815 | 0.0 | 0.534700 |
| 4 | Happy | 0.465295 | 0.607626 | -1.228310 | 0.99726 | 0.482713 | 0.522248 | -1.166696 | 0.999484 | 0.496419 | .. | -0.005616 | 0.0 | 0.538700 | 0.520681 | 0.030072 | 0.0 | 0.543335 |

5 rows x 2005 columns

```
[32]: df.tail()
```

| | class | x1 | y1 | x1 | v1 | x2 | y2 | x2 | v2 | x3 | ... | x499 | v499 | x500 | y500 | x500 | v500 | x501 |
|------|-------|----------|----------|-----------|----------|----------|----------|-----------|----------|----------|-----|-----------|------|----------|----------|----------|------|---------|
| 4890 | Fight | 0.684548 | 0.511105 | -1.037075 | 0.999985 | 0.706993 | 0.431890 | -0.976545 | 0.999968 | 0.722656 | .. | -0.003546 | 0.0 | 0.748149 | 0.430420 | 0.029645 | 0.0 | 0.75333 |
| 4891 | Fight | 0.684453 | 0.511044 | -0.985705 | 0.999985 | 0.706319 | 0.431378 | -0.932847 | 0.999967 | 0.722609 | .. | -0.003345 | 0.0 | 0.747559 | 0.434463 | 0.029979 | 0.0 | 0.75255 |