Data Filtering

class **data_filtering.Outlier_final.OutlierDetection**(*data*) [source]

Bases: `object`

detect_outliers_iqr(*dataset*, *column_names*) [source]

Replaces outliers with NaN using the IQR method.

detect_outliers_isolation_forest(*dataset*, *contamination*, *column_names*) [source]

Replaces outliers with NaN using Isolation Forest.

is_numeric_columns(*dataset*, *column_names*) [source]

Checks if the values in the given columns are numeric.

class **data_filtering.Smoothing_final.SmoothingMethods**(*data*) [source]

Bases: `object`

apply_tes(*data*, *seasonal_periods*, *trend*, *seasonal*, *smoothing_level*, *smoothing_trend*, *smoothing_seasonal*) [source]

Apply Triple Exponential Smoothing (TES) to numeric columns.

calculate_sma(*data*, *window*) [source]

Apply Simple Moving Average (SMA) to numeric columns.

class **data_filtering.Spline_Interpolation_final.SplineInterpolator**(*data*) [source]

Bases: `object`

check_column_validity(*column_name*) [source]

Checks if a column can be interpolated (at least 2 known numeric values).

fill_missing_values() [source]

Applies cubic spline interpolation to fill missing values in numeric columns.

class **data_filtering.Scaling_and_Encoding_final.EncodeAndScaling**(*data*) [source]

Bases: `object`

encode_categorical_features(*feature_data*) [source]

preprocess(*data_object*) [source]

Runs encoding, scaling, and train-test splitting on the dataset.

scale_numerical_features(*encoded_data*) [source]

train_test_split(*processed_data*, *target_column*, *test_size*, *random_state*) [source]

Splits the processed dataset into training and testing sets. :type processed_data: :param processed_data: The encoded and scaled dataset. :type test_size: :param test_size: Proportion of dataset to use as the test set (default 20%). :type random_state: :param random_state: Random seed for reproducibility. :return: Training and testing datasets.

Regression

class **regression.regression_models.RegressionModels** [source]

Bases: `object`

A class to train different regression models including Linear, Polynomial, Ridge, and Lasso regression.

model

The trained regression model (e.g., LinearRegression, Pipeline with PolynomialFeatures and regression).

best_params

The best hyperparameters found during grid search for the respective model (if applicable).

train_linear_regression(*dataobj*) [source]

Trains a Linear Regression model.

train_polynomial_regression(*dataobj*, *param_grid=None*, *cv=5*) [source]

Trains a Polynomial Regression model with grid search.

train_ridge(*dataobj*, *param_grid=None*, *cv=5*) [source]

Trains a Ridge Regression model with polynomial features and grid search.

`train_lasso`(*dataobj*, *param_grid=None*, *cv=5*) [source]

Trains a Lasso Regression model with polynomial features and grid search.

`train_lasso`(*dataobj*, *param_grid=None*, *cv=3*, *subsample_ratio=0.3*) [source]

Train a Lasso Regression model with polynomial features and hyperparameter tuning using GridSearchCV, utilizing a subsample of the training data for efficiency.

Parameters:
- **dataobj** (*dict*) – A dictionary containing split data with keys 'split_data' which further contains 'X_train' and 'y_train' for training features and target variables respectively (expected as pandas DataFrames or Series).
- **param_grid** (*dict*, *optional*) – Dictionary with parameter names (string) as keys and lists of parameters as values, e.g., for polynomial degree and Lasso alpha. Defaults to None.
- **cv** (*int*, *optional*) – Number of cross-validation folds. Defaults to 3.
- **subsample_ratio** (*float*, *optional*) – Fraction of training data to use for hyperparameter tuning. Must be between 0 and 1. Defaults to 0.3.

Returns: The trained Lasso Regression model (best estimator from grid search).

Return type: Pipeline

`self.model`

Stores the trained Lasso Regression model (best estimator).

`self.best_params_lasso`

Stores the best hyperparameters found during grid search.

`self.results_lasso`

Stores the cross-validation results from grid search.

`self.best_degree_lasso`

Stores the best polynomial degree from the best parameters.

Notes

- Subsampling is performed randomly without replacement to reduce computational load.

- The Lasso regression is configured with a maximum of 2000 iterations and a tolerance of 1e-2.

`train_linear_regression`(*dataobj*) [source]

Train a Linear Regression model using the provided training data.

Parameters:	**dataobj** (*dict*) – A dictionary containing split data with keys 'split_data' which further contains 'x_train' and 'y_train' for features and target variables respectively.
Returns:	The trained Linear Regression model.
Return type:	LinearRegression

`self.model`

Stores the trained Linear Regression model.

`train_polynomial_regression`(*dataobj, param_grid=None, cv=5*) [source]

Train a Polynomial Regression model with optional hyperparameter tuning using GridSearchCV.

Parameters:

- **dataobj** (*dict*) – A dictionary containing split data with keys 'split_data' which further contains 'x_train' and 'y_train' for features and target variables respectively.
- **param_grid** (*dict*) – Dictionary with parameters names (string) as keys and lists of parameter as values. Defaults to None.
- **cv** (*int, optional*) – Number of cross-validation folds. Defaults to 5.

Returns: The trained Polynomial Regression model (best estimator from grid search).

Return type: Pipeline

`self.model`

Stores the trained Polynomial Regression model (best estimator).

`self.best_params_poly`

Stores the best hyperparameters found during grid search.

`train_ridge`(*dataobj, param_grid=None, cv=3, subsample_ratio=0.3*) [source]

Train a Ridge Regression model with polynomial features and hyperparameter tuning using GridSearchCV, utilizing a subsample of the training data for efficiency.

Parameters:

- **dataobj** (*dict*) – A dictionary containing split data with keys 'split_data' which further contains 'X_train' and 'y_train' for training features and target variables respectively (expected as pandas DataFrames or Series).
- **param_grid** (*dict*, *optional*) – Dictionary with parameter names (string) as keys and lists of parameters as values, e.g., for polynomial degree and Ridge alpha. Defaults to None.
- **cv** (*int*, *optional*) – Number of cross-validation folds. Defaults to 3.
- **subsample_ratio** (*float*, *optional*) – Fraction of training data to use for hyperparameter tuning. Must be between 0 and 1. Defaults to 0.3.

Returns: The trained Ridge Regression model (best estimator from grid search).

Return type: Pipeline

> **`self.model`**
>
> Stores the trained Ridge Regression model (best estimator).

> **`self.best_params_ridge`**
>
> Stores the best hyperparameters found during grid search.

> **`self.results_ridge`**
>
> Stores the cross-validation results from grid search.

> **`self.best_degree_ridge`**
>
> Stores the best polynomial degree from the best parameters.

Notes

- Subsampling is performed randomly without replacement to reduce computational load.
- The Ridge regression is configured with a maximum of 2000 iterations and a tolerance of 1e-2.

Classification

Module: base_model.py Description: Provides a base class for evaluating classification models.

---

class **`classification.base_model.ClassifierClass`**(*data_train*, *data_test*, *target_train*, *target_test*, *target_labels*) [source]

Bases: `object`

Base class for evaluating classification models.

data_train

Training feature data.

Type: array-like

data_test

Testing feature data.

Type: array-like

target_train

Training target labels.

Type: array-like

target_test

Testing target labels.

Type: array-like

target_labels

List of class labels.

Type: list

display_confusion_matrix(*cm*) [source]

Display the confusion matrix as a percentage plot.

Parameters: **cm** (*np.array*) – The confusion matrix to display.

evaluate(*model*) [source]

Fit the model on training data, predict on test data, and compute evaluation metrics.

Parameters: **model** – A machine learning model instance.
Returns: Contains accuracy, classification report, confusion matrix, and mean squared error.
Return type: tuple

set_model(*model*) [source]

Set the model for evaluation.

Parameters: **model** – A machine learning model instance.

Module: knn_model.py Description: Implements the K-Nearest Neighbors (KNN) classification model with grid search. Best Parameters for KNN: {'n_neighbors': 3, 'p': 1, 'weights': 'uniform'}

---

class **classification.knn_model.KNNModel**(*data_train*, *data_test*, *target_train*, *target_test*, *target_labels*) [source]

Bases: `ClassifierClass`

KNNModel uses K-Nearest Neighbors algorithm for classification and inherits from ClassifierClass.

train() [source]

Train the KNN model using GridSearchCV to determine the best parameters, then prompt the user for final parameter selection and train the model.

Module: random_forest_model.py Description: Implements the Random Forest classification model with grid search parameter tuning. Best Parameters for RandomForest: {'max_depth': 20, 'n_estimators': 150}

---

class **classification.random_forest_model.RandomForestModel**(*data_train*, *data_test*, *target_train*, *target_test*, *target_labels*) [source]

Bases: `ClassifierClass`

RandomForestModel uses a Random Forest algorithm for classification and inherits from ClassifierClass.

train() [source]

Train the Random Forest model using GridSearchCV to determine the best parameters, then prompt the user for final parameter selection and train the model.

Module: svc_model.py Description: Implements the Support Vector Classifier (SVC) model with grid search parameter tuning. Best Parameters for SVC: {'C': 1, 'gamma': 'scale', 'kernel': 'rbf'}

---

class **classification.svc_model.SVCModel**(*data_train*, *data_test*, *target_train*, *target_test*, *target_labels*) [source]

Bases: `ClassifierClass`

SVCModel uses the Support Vector Classifier for classification and inherits from ClassifierClass.

train() [source]

Train the SVC model using GridSearchCV to determine the best parameters, then prompt the user for final parameter selection and train the model.

AI Module

class **ai_model.ann.ArtificialNeuralNetwork**(*problem_type='classification', options=None*) [source]

Bases: `BaseModel`

Implements an Artificial Neural Network (ANN) model for classification.

model

The ANN model instance.

Type: Sequential

problem_type

Type of problem (only “classification” is supported).

Type: str

batch_size

Number of samples per batch during training.

Type: int

epochs

Number of training iterations.

Type: int

evaluate() [source]

Evaluates the trained model on the test data.

Returns: A dictionary containing evaluation metrics: - `Accuracy` (float): Accuracy score of the model. - `Confusion Matrix` (list): Confusion matrix representing classification performance.

Return type: dict

Raises: **ValueError** – If test data is missing.

load_weights(*filepath='ann_weights.h5'*) [source]

Loads pre-trained model weights from a file.

Parameters: **filepath** (*str, optional*) – Path from where the weights will be loaded (default is “ann_weights.h5”).

Raises:
- **FileNotFoundError** – If the specified file does not exist.
- **Exception** – If an error occurs while loading weights.

`save_weights`(*filepath='ann_weights.h5'*) [source]

Saves the trained model’s weights to a file.

Parameters: **filepath** (*str, optional*) – Path where the weights will be saved (default is “ann_weights.h5”).

Raises: **Exception** – If an error occurs while saving weights.

`train`() [source]

Trains the ANN model using the provided training data.

Raises: **ValueError** – If training data is missing.

---

class `ai_model.base.BaseModel`(*model, problem_type='classification'*) [source]

Bases: `object`

Base class for all models that centralizes hyperparameter validation and common functionalities like data splitting, training, and evaluation.

`HYPERPARAMETER_RANGES`

Defines the valid range of hyperparameters for different model types.

Type: dict

`model`

The machine learning model instance.

Type: object

`problem_type`

Type of problem (either “classification” or “regression”).

Type: str

`X_train`

Training feature dataset.

Type: pandas.DataFrame or None

X_test

Testing feature dataset.

Type: pandas.DataFrame or None

y_train

Training target dataset.

Type: pandas.Series or None

y_test

Testing target dataset.

Type: pandas.Series or None

HYPERPARAMETER_RANGES
= {'ArtificialNeuralNetwork': {'activation': {'allowed': ['relu', 'sigmoid', 'tanh', 'softmax'], 'default': ['relu', 'relu', 'softmax']}, 'batch_size': {'default': 30, 'max': 128, 'min': 16}, 'epochs': {'default': 100, 'max': 300, 'min': 10}, 'layer_number': {'default': 3, 'max': 6, 'min': 1}, 'optimizer': {'allowed': ['adam', 'sgd', 'rmsprop'], 'default': 'adam'}, 'units': {'default': [128, 64, 4], 'max': 256, 'min': 1}}, 'CatBoost': {'learning_rate': {'default': 0.03, 'max': 0.1, 'min': 0.01}, 'max_depth': {'default': 6, 'max': 10, 'min': 4}, 'n_estimators': {'default': 500, 'max': 1000, 'min': 100}, 'reg_lambda': {'default': 3, 'max': 10, 'min': 1}}, 'RandomForest': {'max_depth': {'default': 20, 'max': 50, 'min': 3}, 'min_samples_leaf': {'default': 1, 'max': 10, 'min': 1}, 'min_samples_split': {'default': 5, 'max': 10, 'min': 4}, 'n_estimators': {'default': 200, 'max': 500, 'min': 10}}, 'XGBoost': {'learning_rate': {'default': 0.3, 'max': 0.3, 'min': 0.01}, 'max_depth': {'default': 6, 'max': 10, 'min': 0}, 'min_split_loss': {'default': 10, 'max': 10, 'min': 3}, 'n_estimators': {'default': 200, 'max': 1000, 'min': 100}}}

static **validate_options**(*options*, *model_type*) [source]

Validates and ensures that the provided hyperparameters fall within allowed ranges.

Parameters:
- **options** (*dict*) – The dictionary containing model hyperparameters.
- **model_type** (*str*) – The type of model being validated.

Returns: A dictionary of validated hyperparameters with out-of-range values replaced with defaults.

Return type: dict

Raises: **Exception** – If an unexpected error occurs during validation.

---

class **ai_model.catboost_model.Catboost**(*problem_type='classification'*, *options=None*) [source]

Bases: `BaseModel`

Implements the CatBoost model for both classification and regression, ensuring hyperparameters are validated before model initialization.

`problem_type`

Specifies whether the model is for classification or regression.

Type: str

`options`

Contains hyperparameters such as *n_estimators*, *learning_rate*, *max_depth*, and *reg_lambda*.

Type: dict

---

class **ai_model.xgboost_model.XGBoost**(*problem_type='regression', options=None*) [source]

Bases: `BaseModel`

This module provides an implementation of the XGBoost model for regression. It extends the BaseModel class and ensures that hyperparameters are validated before model initialization.

A class to implement the XGBoost model for regression.

Parameters:
- **problem_type** – Defines the type of problem being solved (only regression supported).
- **options** – Contains hyperparameters such as *n_estimators*, *learning_rate*, *min_split_loss*, and *max_depth*.

Raises:
- **ValueError** – If *problem_type* is not "regression".
- **Exception** – If an error occurs during model initialization.

---

class **ai_model.random_forest.RandomForest**(*problem_type='classification', options=None*) [source]

Bases: `BaseModel`

A class to implement the Random Forest model for both classification and regression.

Attributes:

problem_type : str

Defines whether the model is for classification or regression.

options : dict

Contains hyperparameters such as n_estimators, max_depth, min_samples_split, and min_samples_leaf.

Image Processing

class **image_processing.dataloader.DataLoadingAndPreprocessing**(*image_size=(28, 28)*) [source]

Bases: `object`

A class to handle data loading and preprocessing.

image_size

Size of images (default is (28, 28)).

Type: tuple

data

Loaded data as a Pandas DataFrame.

Type: DataFrame

labels

List of labels.

Type: list

label_dict

Dictionary mapping labels to integers.

Type: dict

images

List of processed images.

Type: list

data_loader(*dataset_name*, *is_zipped=True*) [source]

Loads and preprocesses image dataset.

get_label_dict() [source]

Returns label dictionary.

encode_labels() [source]

Encodes labels into numeric values.

create_labels(*folder_name*, *is_zipped=True*) [source]

Creates and encodes labels from directory structure.

unzip_folder(*current_path*, *folder_name*, *data_dir*) [source]

Unzips dataset if necessary.

normalize_dataset() [source]

Normalizes dataset to the range [0,1].

split_dataset(*test_size=0.2*, *random_state=42*) [source]

Splits dataset into training and test sets.

create_labels(*data_dir*) [source]

Creates and encodes labels from the dataset folder.

This method assumes that the dataset consists of subdirectories named after the class labels, each containing images of that class. It processes these subdirectories, encodes the labels numerically, and stores them in a DataFrame.

Parameters: **data_dir** (*str*) – The name of the folder containing the dataset.
Returns: None

data_loader(*dataObj*) [source]

This method loads the data from the dataset folder (zipped or unzipped), creates labels, encodes them, loads the dataset and normalizes the dataset.

Parameters: dataObj: dict

A data object dictionary containing

dataset_name: str

The name of the folder where the images are stored

is_zipped: bool

If the dataset is zipped or not

encode_labels() [source]

Encodes the labels into numerical values.

This method assigns a unique integer to each label and maps the dataset labels to their corresponding numerical values.

Returns: None

`get_label_dict`() [source]

Retrieves the label dictionary.

Returns: Mapping of label names to integers.
Return type: dict

`normalize_dataset`() [source]

Normalizes the dataset by converting images to numpy arrays and scaling pixel values.

This method ensures that pixel values are in the range [0,1] for better training efficiency in deep learning models.

Returns: None

`split_dataset`(*dataObj*) [source]

Splits the dataset into training and testing sets.

This method partitions the preprocessed dataset into training and test sets, ensuring reproducibility with a fixed random state.

Parameters: **dataObj** (*dict*) – The data object dictionary consisting of: test_size (float): The proportion of the dataset to include in the test split. Defaults to 0.2. random_state (int): The seed used by the random number generator for reproducibility. Defaults to 42.

Returns:

A tuple containing four numpy arrays:

- X_train (numpy.ndarray): Training images.
- y_train (numpy.ndarray): Training labels.
- X_test (numpy.ndarray): Test images.
- y_test (numpy.ndarray): Test labels.

Return type: tuple

`unzip_folder`(*current_path*, *zip_file*, *data_dir*) [source]

Extracts a ZIP file if it is not already unzipped.

This method checks if the dataset directory exists. If not, it attempts to extract the ZIP file containing the dataset.

Parameters:

- **current_path** (*str*) – The path where the script is executed.
- **zip_file** (*str*) – The name of the zipfile.
- **data_dir** (*str*) – The directory where the dataset should be extracted.

Returns: None

class **image_processing.evaluator.Evaluation**(*model*) [source]

Bases: `object`

A class for evaluating a trained model and visualizing its performance using a confusion matrix.

evaluate_model(*dataObj*) [source]

Evaluates the model on the test dataset.

Attributes:

X_test : numpy.ndarray

The test dataset features.

y_test : numpy.ndarray

The true labels for the test dataset.

Returns:

: - test_acc: Test accuracy. - test_loss: Test loss.

get_confusion_matrix(*dataObj*, *pred_tuple*) [source]

Generates the confusion matrix.

Parameters:

pred_tuple : tuple

A tuple containing the predicted labels and indicies for the test dataset.

Returns:

: dict: A dictionary containing six key value pairs:

- labels: Unique labels from the dataset
- values: The confusion matrix values in percentage
- xlabel: X-axis label to be used for visualization.
- ylabel: Y-axis label to be used for visualization.
- title: Graph title to be used for visualization.
- tick_marks: Tick marks to be used for visualization.

class **image_processing.nn.NeuralNetwork** [source]

Bases: `object`

A class to create and manage a Convolutional Neural Network (CNN) model using TensorFlow and Keras.

classmethod **create_cnn_model**(*dataObj*) [source]

Create a Convolutional Neural Network (CNN) model with configurable optimizer, loss function, and activation functions.

The model consists of: - Two convolutional layers with ReLU activation. - Two max-pooling layers. - A fully connected dense layer with 128 neurons and ReLU activation. - An output layer with 10 neurons and softmax activation for multi-class classification.

Parameters:	**dataObj** (*dict*) – Data dictionary containing: - optimizer (str): Optimizer to compile the model (e.g., 'adam', 'RMSPROP' & 'adamax'). - activation_function (str): Activation function for hidden layers (e.g., 'relu', 'sigmoid').
Returns:	A compiled CNN model.
Return type:	model

class **image_processing.test.Testing**(*model*, *dataObj*) [source]

Bases: `object`

A class for testing a trained model by making predictions and visualizing results.

model

The trained machine learning model.

Type: object

X_test

The test dataset features.

Type: numpy.ndarray

y_test

The test dataset labels.

Type: numpy.ndarray

label_dict

A dictionary mapping class indices to class labels.

Type: dict, optional

set_label_dict(*label_dict*) [source]

Stores a reverse mapping of the label dictionary.

make_predictions() [source]

Generates predictions using the trained model.

plot_image(*pred_tuple*, *index*) [source]

Displays a sample test image along with its predicted and true labels.

get_predicted_tuple() [source]

Converts model output to class labels and probabilities.

get_predicted_tuple() [source]

Converts model predictions to a tuple of (index, predicted label, probability).

Returns: Each tuple contains (predicted class index, predicted class name, prediction probability).

Return type: list of tuples

make_predictions(*X_test_reshaped*) [source]

Makes predictions using the trained model.

Returns: The predicted output from the model.

Return type: numpy.ndarray

plot_image(*pred_tuple*, *index*) [source]

Visualizes a sample image and shows predictions.

Parameters:

- **pred_tuple** (*list of tuples*) – A list of tuples containing (predicted index, predicted label, predicted probability).
- **index** (*int*) – The index of the image in the test dataset.

Raises: **ValueError** – If the label dictionary is not set.

set_label_dict(*label_dict*) [source]

Stores the reverse mapping of a label dictionary.

Parameters: **label_dict** (*dict*) – A dictionary mapping class names to class indices.

class `image_processing.train.Training`(*model*) [source]

Bases: `object`

A class for testing a trained model by making predictions and visualizing results.

Attributes:

model : object

The trained machine learning model.

X_test : numpy.ndarray

The test dataset features.

y_test : numpy.ndarray

The test dataset labels.

Methods:

make_predictions(use_cnn=False):

Generates predictions using the trained model.

plot_image(index, use_cnn=False):

Displays a sample test image along with its predicted and true labels.

get_predicted_labels(y_predicted):

Converts model output to class labels.

`train_nn`(*dataObj*, *epochs=10*) [source]

Trains the neural network model on the training data.

Parameters:

dataObj : dict

Data object dictionary containing training and test datasets (X_train, y_train, X_test, y_test).

epochs : int, optional

Number of training epochs (default is 10).