Deltares

Datafusiontools

Development framework

Eleni Smyrniou

Bruno Zuada Coelho

28/1/2022

Data fusion tools

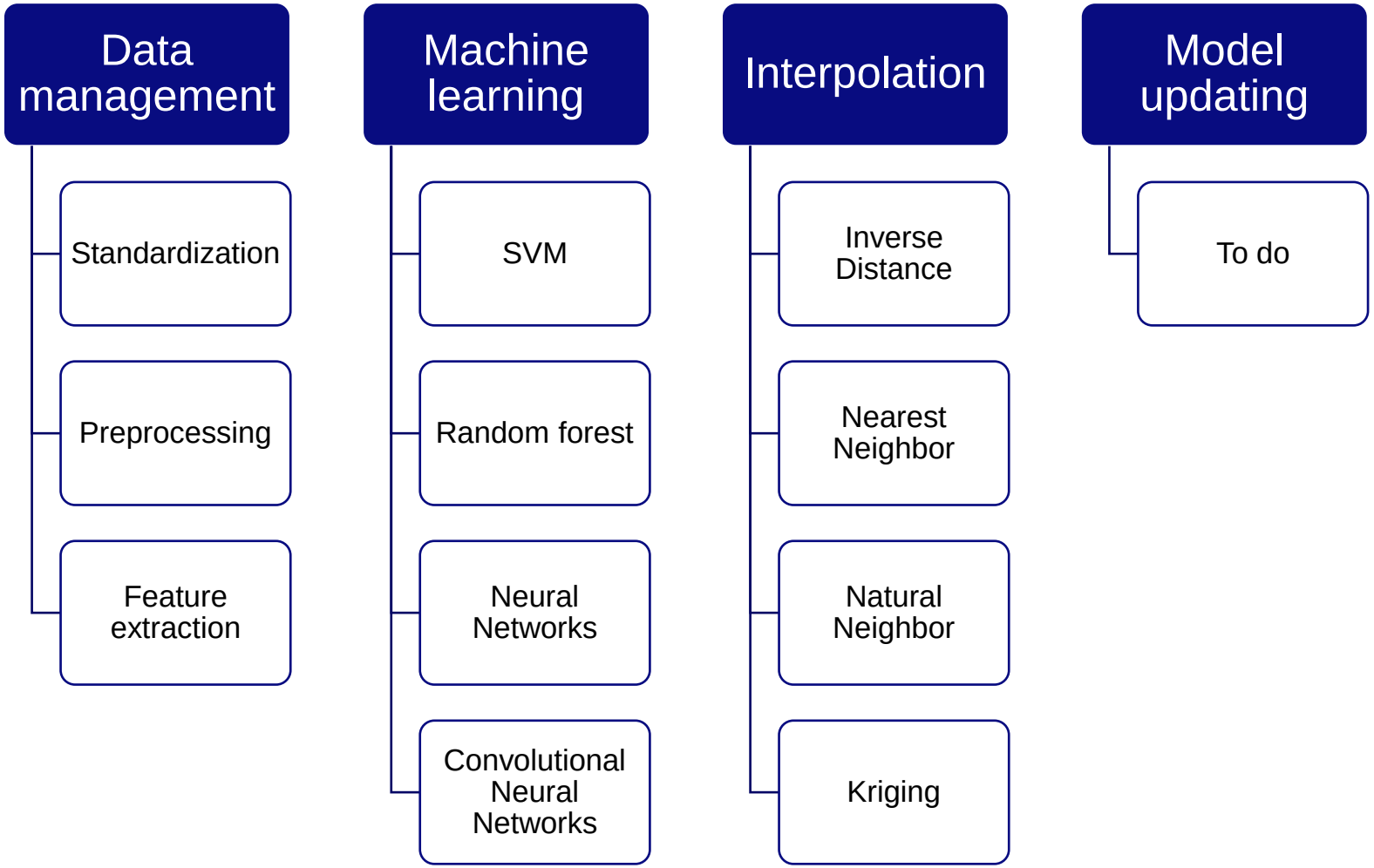
- Standardize input for machine learning algorithms and interpolation
- Create utils that facilitate creation of feature/targets
- Developers/users can create their own models
- Basic existing models included

Data fusion tools

- Project in python:
  - Several libraries for ML / data analysis
  - Large community
- Development in bitbucket:
  - Revision control system
  - Trello for issue tracking
  - Pipelines for unit and integration testing
  - Automatic documentation
  - Automatic packaging
- Send us your e-mail to add you to the bitbucket!
  - https://bitbucket.org/DeltaresGEO/datafusiontools/src/master/
- To use:
  - `pip install git+https://bitbucket.org/DeltaresGEO/datafusiontools/src/master`

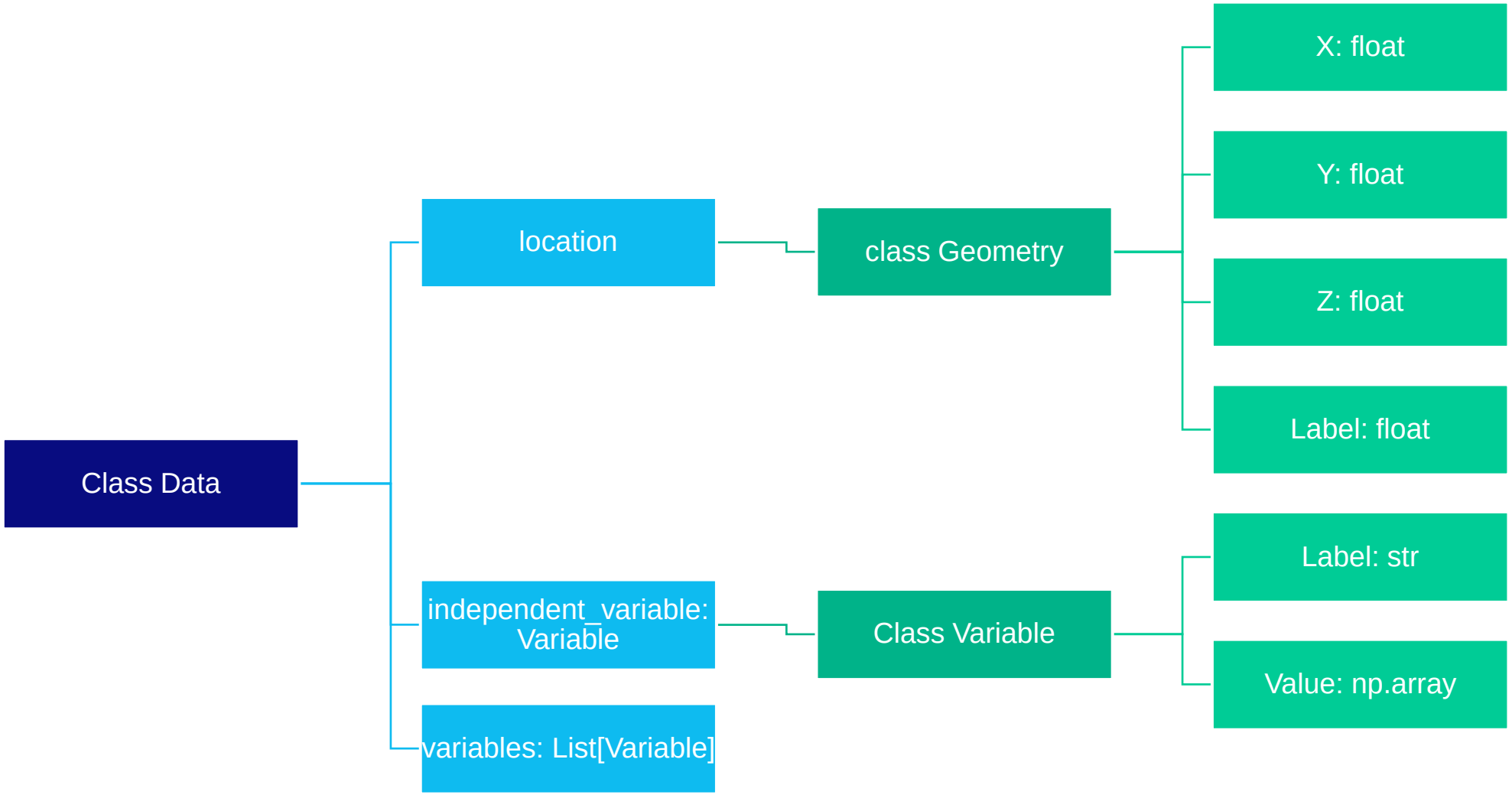
  or
  - `pip install DataFusionTools-0.1-py3-none-any.whl`

Data fusion tools modules



Data management module

Data management – standardization

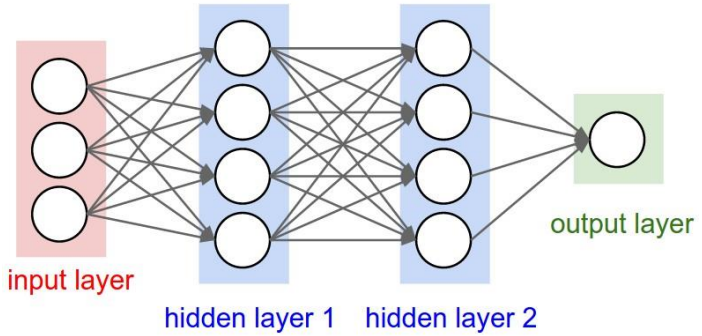


Data management – preprocessing and feature extraction

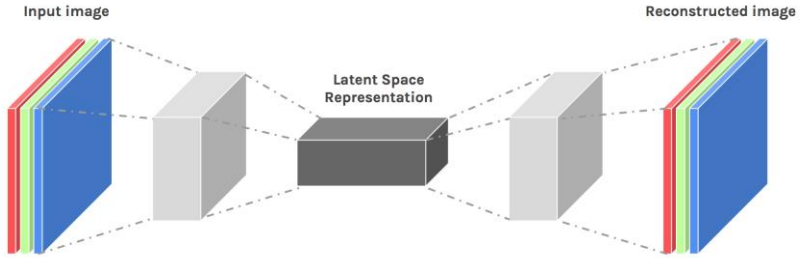
CreateInputsML class	Split train-test-validation data
	Add features/targets from different data classes
	Link different data classes based on their location
	Get feature/target arrays for the algorithms

Machine learning module

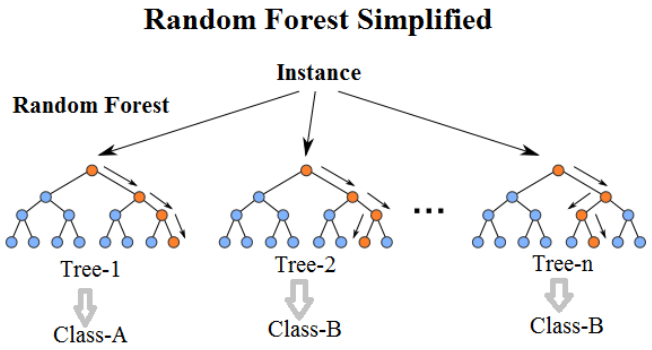
Machine learning models included in datafusion tools



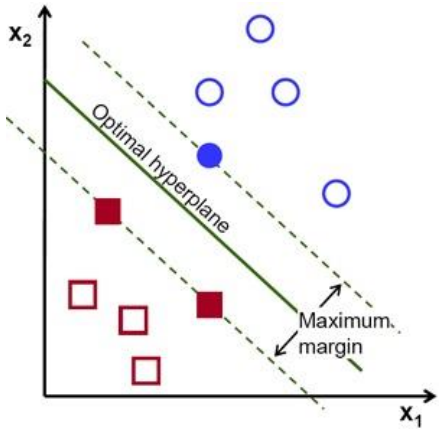
Neural network



Convolutional neural network

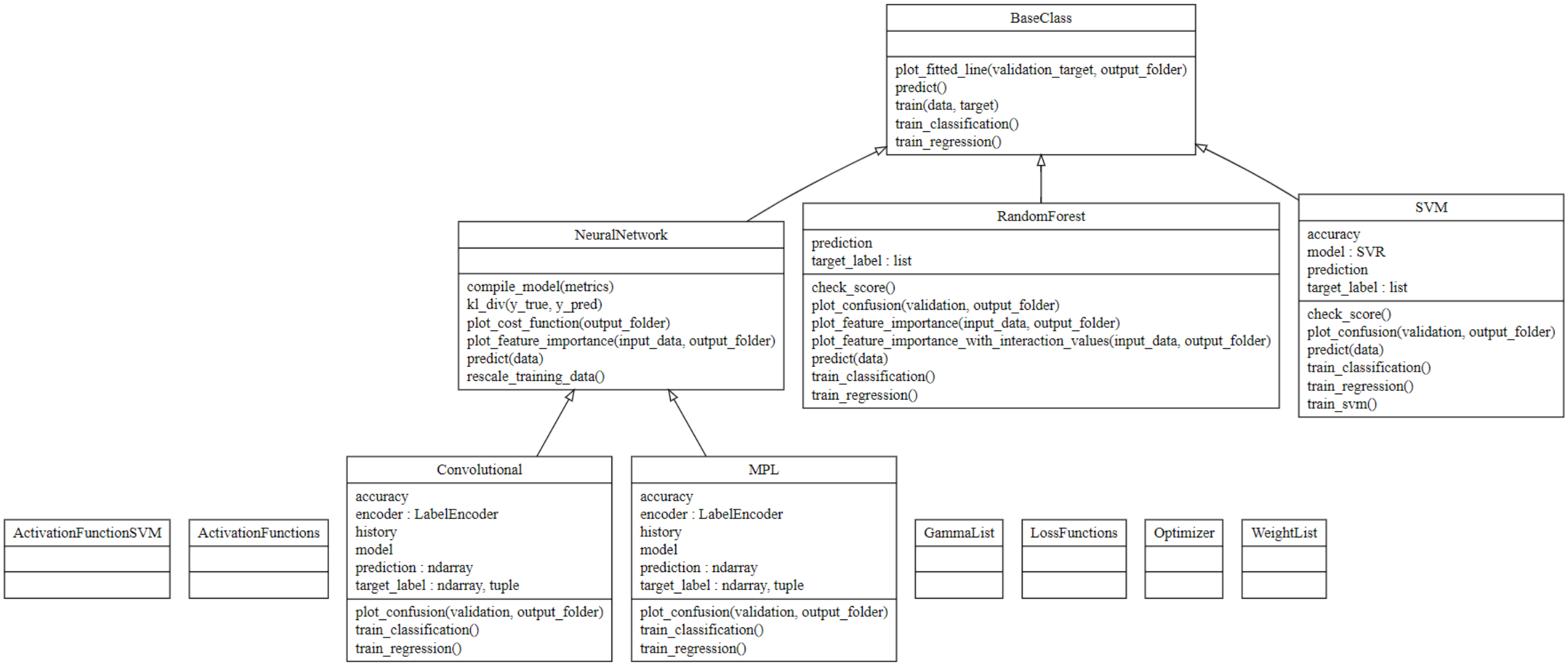


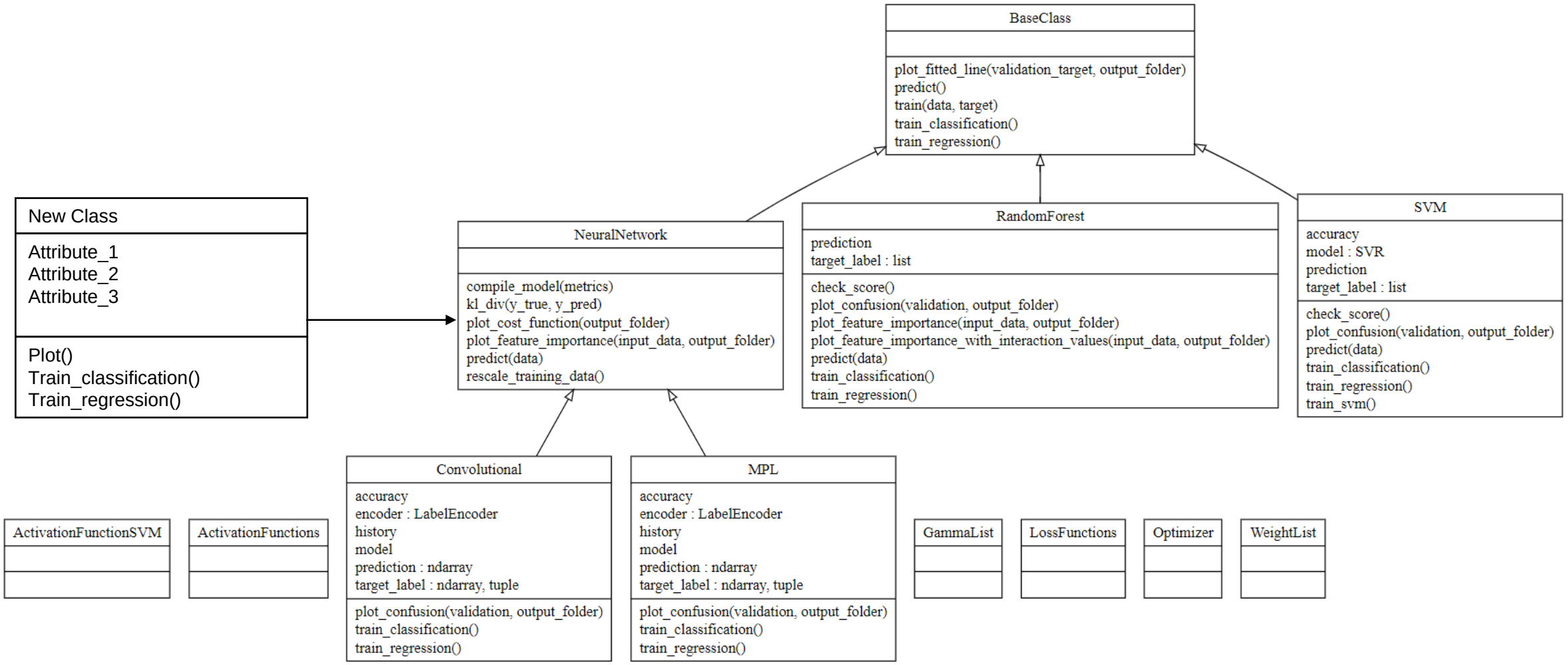
Random forest



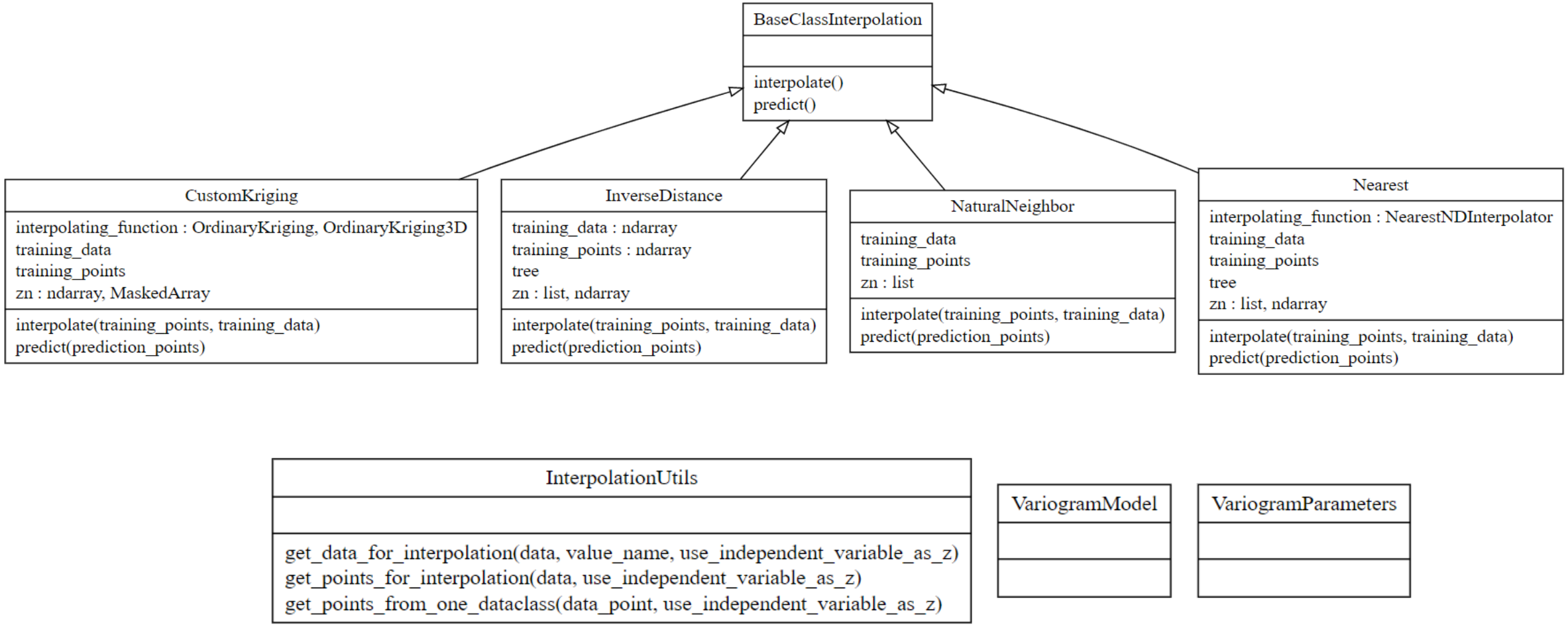
Support vector machine

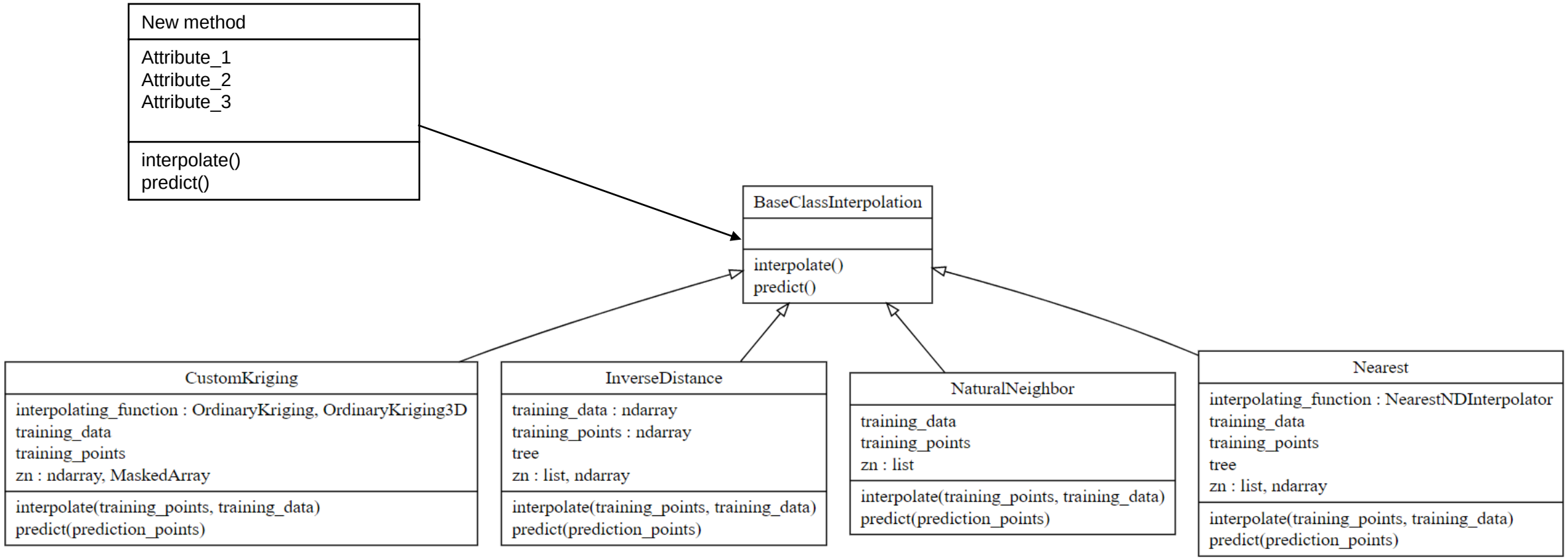
Deltares


BaseClass
plot_fitted_line(validation_target, output_folder)
predict()
train(data, target)
train_classification()
train_regression()
NeuralNetwork
compile_model(metrics)
kl_div(y_true, y_pred)
plot_cost_function(output_folder)
plot_feature_importance(input_data, output_folder)
predict(data)
rescale_training_data()
RandomForest
prediction
target_label : list
check_score()
plot_confusion(validation, output_folder)
plot_feature_importance(input_data, output_folder)
plot_feature_importance_with_interaction_values(input_data, output_folder)
predict(data)
train_classification()
train_regression()
SVM
accuracy
model : SVR
prediction
target_label : list
check_score()
plot_confusion(validation, output_folder)
predict(data)
train_classification()
train_regression()
train_svm()
Convolutional
accuracy
encoder : LabelEncoder
history
model
prediction : ndarray
target_label : ndarray, tuple
plot_confusion(validation, output_folder)
train_classification()
train_regression()
MPL
accuracy
encoder : LabelEncoder
history
model
prediction : ndarray
target_label : ndarray, tuple
plot_confusion(validation, output_folder)
train_classification()
train_regression()
ActivationFunctionSVM
ActivationFunctions
GammaList
LossFunctions
Optimizer
WeightList


BaseClass
plot_fitted_line(validation_target, output_folder)
predict()
train(data, target)
train_classification()
train_regression()
New Class
Attribute_1
Attribute_2
Attribute_3
Plot()
Train_classification()
Train_regression()
NeuralNetwork
compile_model(metrics)
kl_div(y_true, y_pred)
plot_cost_function(output_folder)
plot_feature_importance(input_data, output_folder)
predict(data)
rescale_training_data()
RandomForest
prediction
target_label : list
check_score()
plot_confusion(validation, output_folder)
plot_feature_importance(input_data, output_folder)
plot_feature_importance_with_interaction_values(input_data, output_folder)
predict(data)
train_classification()
train_regression()
SVM
accuracy
model : SVR
prediction
target_label : list
check_score()
plot_confusion(validation, output_folder)
predict(data)
train_classification()
train_regression()
train_svm()
Convolutional
accuracy
encoder : LabelEncoder
history
model
prediction : ndarray
target_label : ndarray, tuple
plot_confusion(validation, output_folder)
train_classification()
train_regression()
MPL
accuracy
encoder : LabelEncoder
history
model
prediction : ndarray
target_label : ndarray, tuple
plot_confusion(validation, output_folder)
train_classification()
train_regression()
ActivationFunctionSVM
ActivationFunctions
GammaList
LossFunctions
Optimizer
WeightList

Interpolation module


BaseClassInterpolation
interpolate()
predict()
CustomKriging
interpolating_function : OrdinaryKriging, OrdinaryKriging3D
training_data
training_points
zn : ndarray, MaskedArray
interpolate(training_points, training_data)
predict(prediction_points)
InverseDistance
training_data : ndarray
training_points : ndarray
tree
zn : list, ndarray
interpolate(training_points, training_data)
predict(prediction_points)
NaturalNeighbor
training_data
training_points
zn : list
interpolate(training_points, training_data)
predict(prediction_points)
Nearest
interpolating_function : NearestNDInterpolator
training_data
training_points
tree
zn : list, ndarray
interpolate(training_points, training_data)
predict(prediction_points)
InterpolationUtils
get_data_for_interpolation(data, value_name, use_independent_variable_as_z)
get_points_for_interpolation(data, use_independent_variable_as_z)
get_points_from_one_dataclass(data_point, use_independent_variable_as_z)
VariogramModel
VariogramParameters


New method
Attribute_1
Attribute_2
Attribute_3
interpolate()
predict()
BaseClassInterpolation
interpolate()
predict()
CustomKriging
interpolating_function : OrdinaryKriging, OrdinaryKriging3D
training_data
training_points
zn : ndarray, MaskedArray
interpolate(training_points, training_data)
predict(prediction_points)
InverseDistance
training_data : ndarray
training_points : ndarray
tree
zn : list, ndarray
interpolate(training_points, training_data)
predict(prediction_points)
NaturalNeighbor
training_data
training_points
zn : list
interpolate(training_points, training_data)
predict(prediction_points)
Nearest
interpolating_function : NearestNDInterpolator
training_data
training_points
tree
zn : list, ndarray
interpolate(training_points, training_data)
predict(prediction_points)

Example

User workflow – example

Insert data in dataclasses

- Define/organize/group data
- Cpts/ insar / resistivity data are grouped together based on how the user wants to organize them

Features and targets are created using utils class

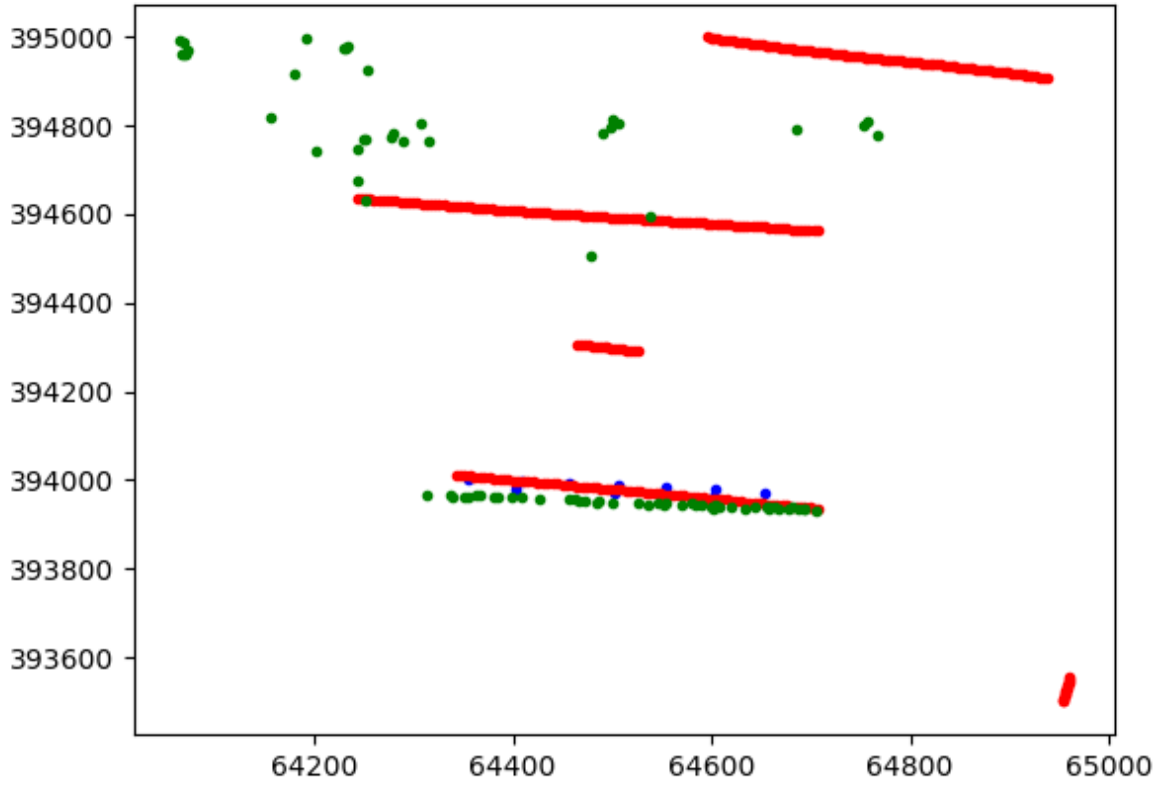
- Define features and targets for machine learning model
- Resistivity and insar are selected as inputs while IC is the output

Run model and evaluate the results

- Evaluate using test data
- Create plots

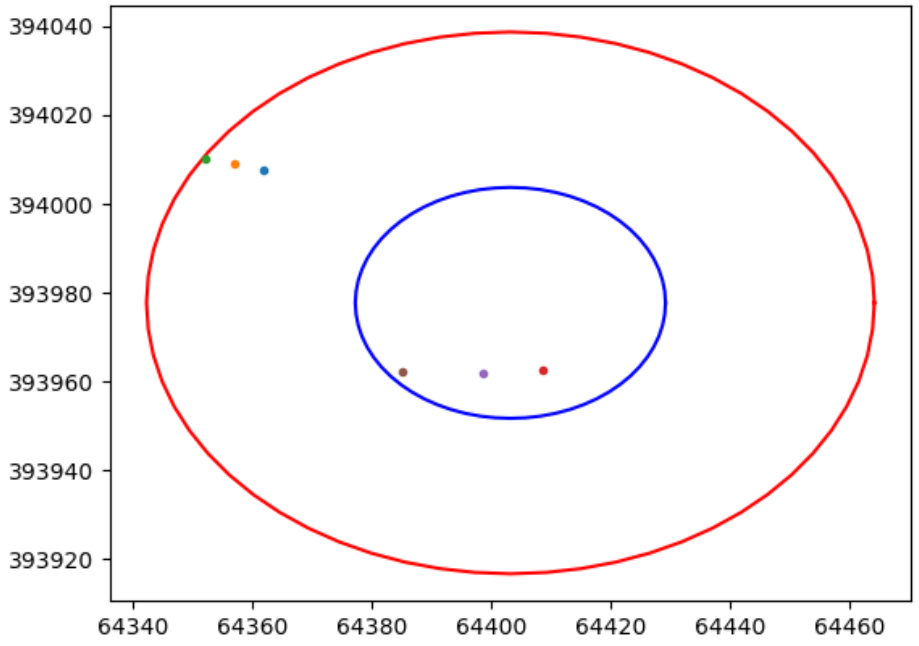
Data example

- Data
  - Cpt
  - Insar
  - Resistivity
- User should pre-process the data so that they arranged per cpt point
  - Searching for the closest resistivity and insar points

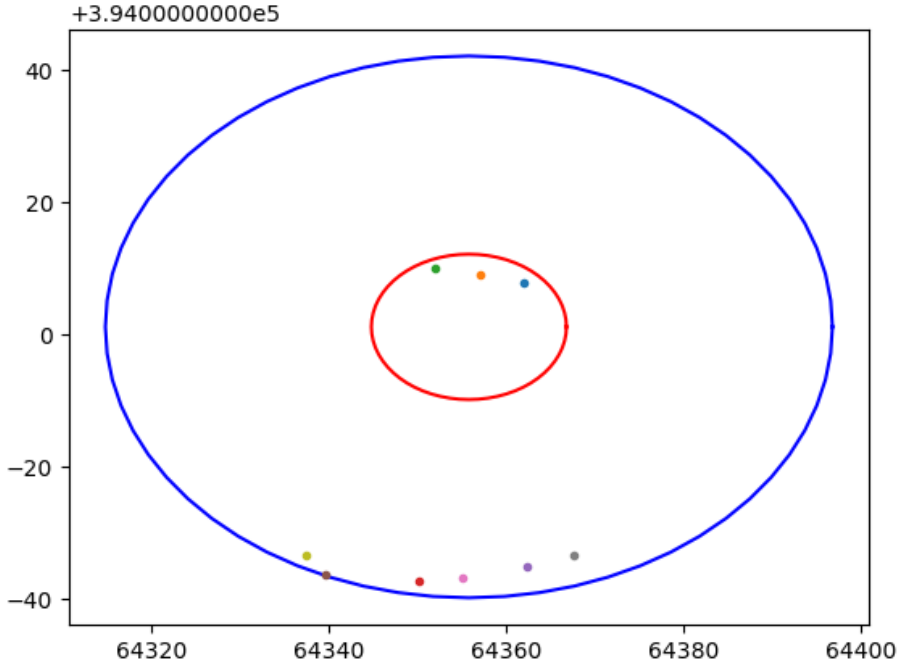


Data – grouping resistivity and insar points

CPT000000092599

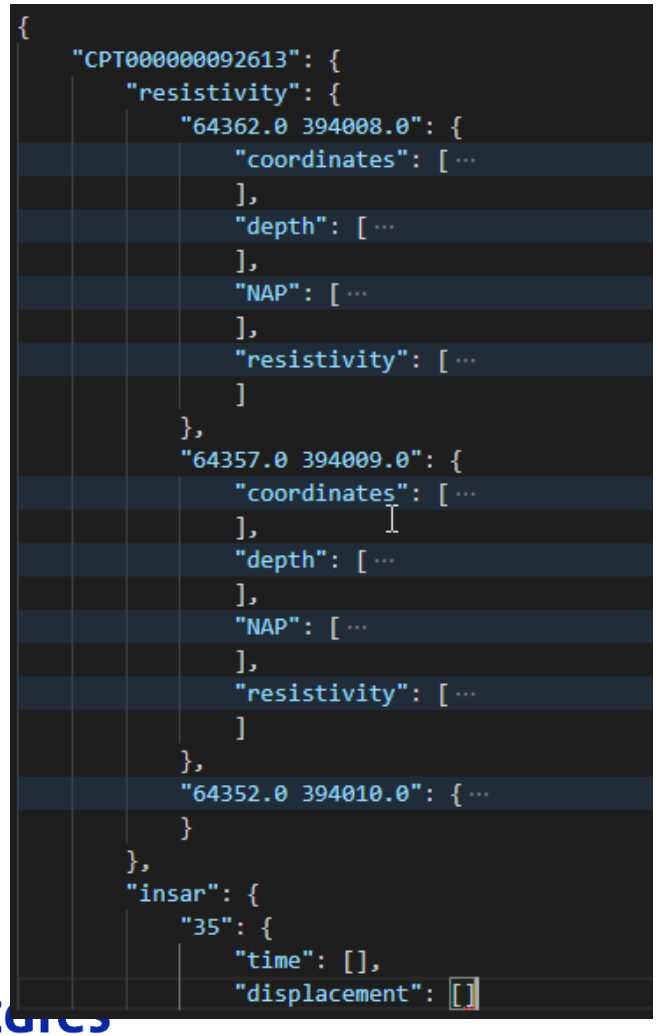


CPT000000092663



Deltares

Grouped data



Resistivity measurement point 1

Resistivity measurement point 2

Insar data point 1

Deltares

Collect data class example

```
# create input class
data_cpts = []
for name, item in combined.items():
    cpt = cpts.get(name)
    depth = Variable(label="depth", value=np.array(cpt.get("depth")))
    tip = Variable(label="tip", value=np.array(cpt.get("tip")))
    IC = Variable(label="IC", value=np.array(cpt.get("IC")))
    lithology = Variable(
        label="lithology", value=np.array(cpt.get("lithology"))
    )
    cpt_new_data_structure = Data(
            location=Geometry(
                x=cpt["coordinates"][0], y=cpt["coordinates"][1], z=0
            ),
            variables=[tip, IC, lithology, resistivity],
            independent_variable=depth,
        )
    data_cpts.append[cpt_new_data_structure]
```

Variables are collected and inputted into our class Variable

Data class is initialized with all the data collected. This contains one cpt

All data collected in a list

Create features and targets

```
create_features = CreateInputsML()
for name, data in data_cpts.items():
    create_features.add_features(
        data["cpt"],
        ["lithology"],
        use_independent_variable=True,
        use_location_as_input=True,
    )
    create_features.add_targets(data["cpt"], ["IC"])
```

- Functionality of CreateInputsML:
  - User can add features for training
    - User can specify which variables they want to use based on their names
    - Independent variable (depth) can be used as feature
    - Location can be used as feature
  - User can specify targets
  - Split train/test/validation data

Linking features based on their location

CPT at location x1	
depth	IC
0	2.05
-0.1	2.08
-0.2	2.11
-0.3	2.14
-0.4	2.17
-0.5	2.2

Insar at location x2	
time	insar value
1	0
2	0.0035

Link location x1
with location x2

Linked features			
depth	IC	insar value_1	insar value_2
0	2.05	0	0.0035
-0.1	2.08	0	0.0035
-0.2	2.11	0	0.0035
-0.3	2.14	0	0.0035
-0.4	2.17	0	0.0035
-0.5	2.2	0	0.0035

Linking features based on their location

```
    for name, data in data_collection.items():
        row_records_in_geometry.append(data["cpt"].location)
        column_records_in_geometry.append(data["insar_rate"].location)
     create_features.link_geometries(
        row_records_in_geometry=row_records_in_geometry,
        column_records_in_geometry=column_records_in_geometry,
     )
```

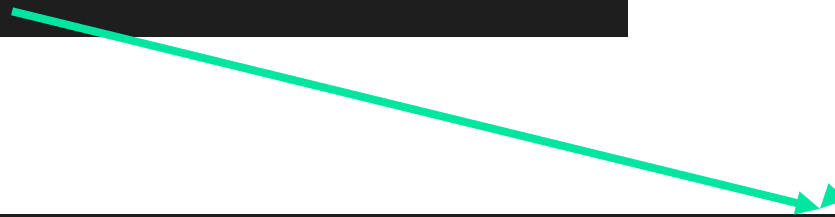
Create and train models

Neural Network

```
model = MPL(
    classification=False,
    nb_hidden_layers=3,
    nb_neurons=[18, 18, 14],
    activation_fct=class_enums.ActivationFunctions.relu,
    optimizer=class_enums.Optimizer.Adam,
    loss=class_enums.LossFunctions.mean_squared_error,
    epochs=50,
    batch=1,
    regularisation=0,
    feature_names=feature_names,
    validation_features=extra_validation_training,
    validation_targets=extra_validation_target,
)
```

Random Forest

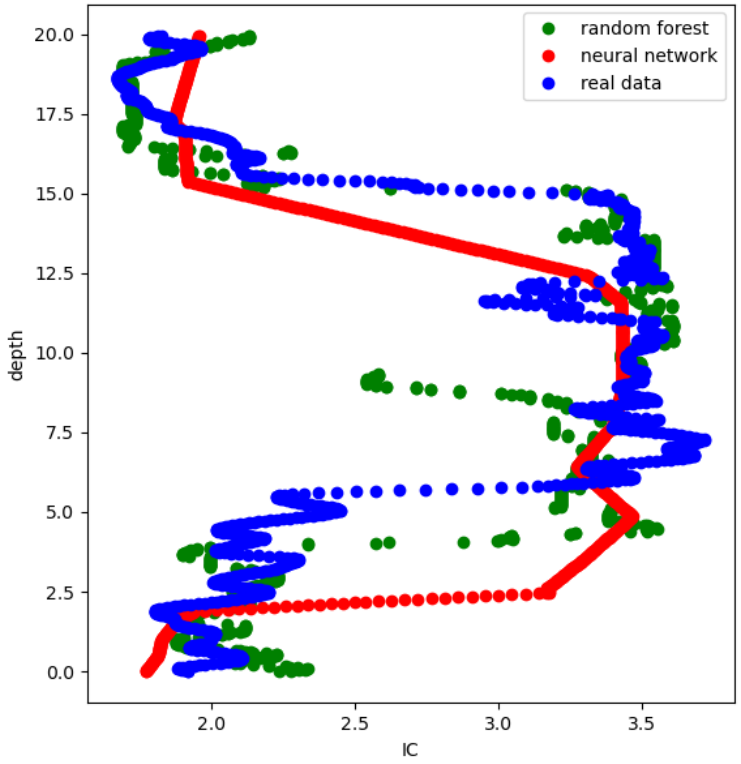
```
model = RandomForest(
    classification=False,
    n_estimator=np.linspace(1, 10, 10),
    max_depth=np.linspace(1, 50, 50),
    feature_names=features_names,
)
```



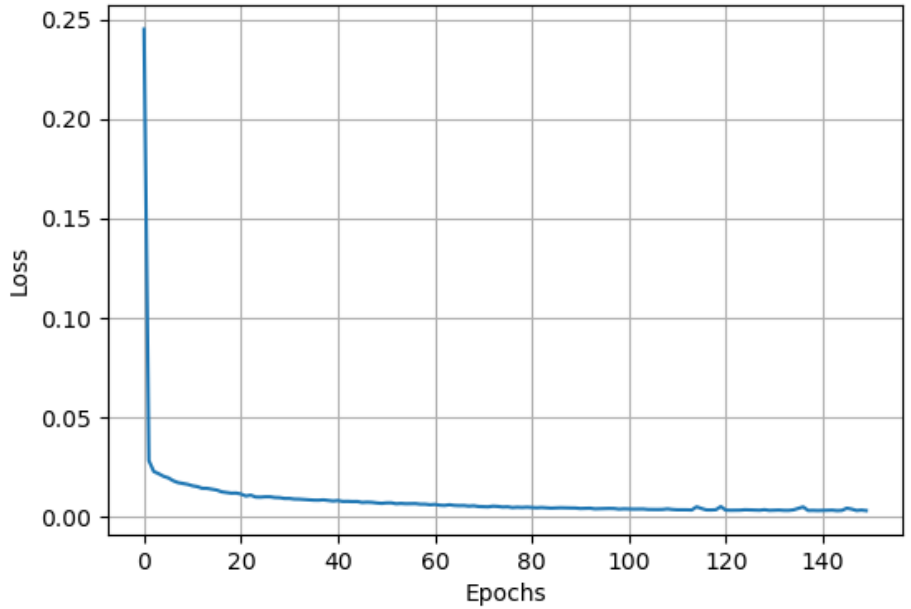
```
model.train(
    training_data,
    target_data,
)
model.predict(validation_training)
model.plot_feature_importance(validation_training, Path("./tests/test_output"))
model.plot_fitted_line(validation_target, Path("./tests/test_output"))
```

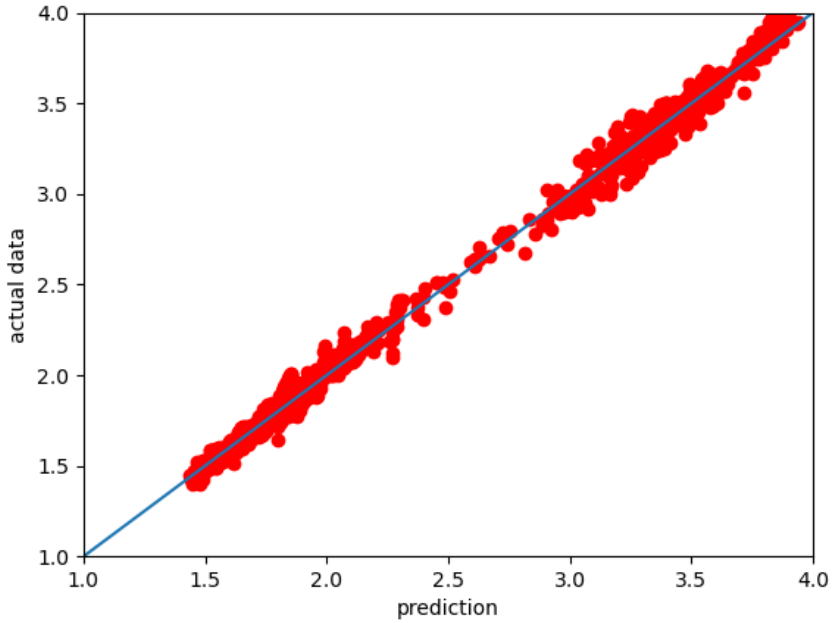
Deltares

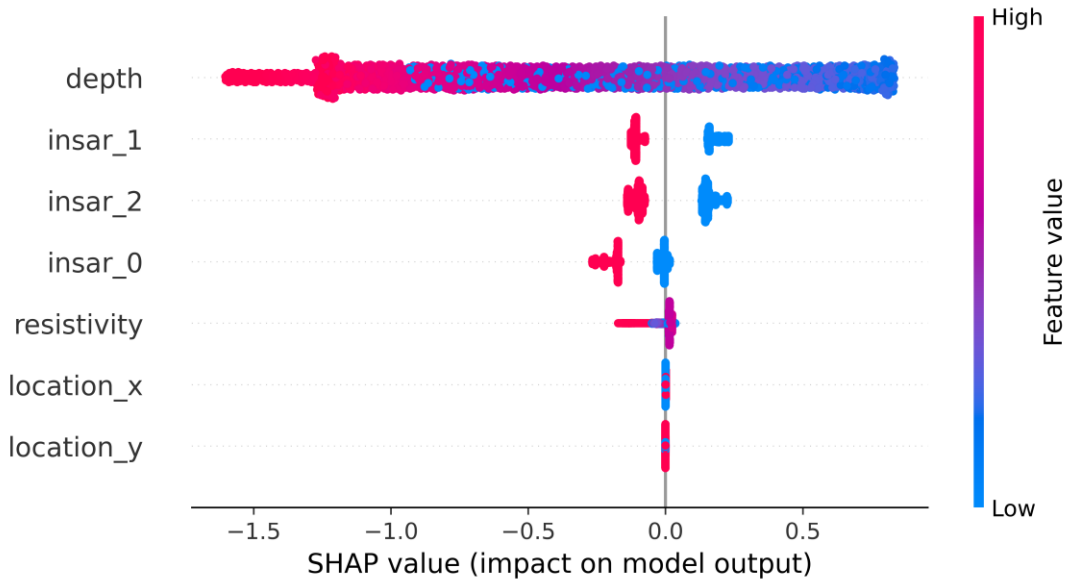
Comparing different models



Plots for evaluating the results







Deltares

How to contribute

- Provide your email address!
- Contribute with new methods for existing modules:
  - A new ML algorithm
  - A new interpolation technique
- Develop new methods:
  - Parsers to data & visualisations
- Use the package on your projects
  - At the moment the package is only available for the consortium. The package will be made publicly available (with a license)

Contact

www.deltares.nl

@deltares

linkedin.com/company/deltares

info@deltares.nl

@deltares

facebook.com/deltaresNL