# FastML

***Release 1.0***

**clem**

Nov 05, 2021

Table of Contents

Chapter 1

# FastML

## 1.1 The class Model

The model needs only two lines to be created, fitted, tested and saved
Another line to be optimized
Another line for inference

**class** api.**Model** ( *model: callable, X_train, X_test, y_train, y_test, parameters={}, metrics={}, name=''* )

Model ML

**Parameters**

- **model** (*callable*) – the model to use
- **X_train** (*array*) – the train data
- **X_test** (*array*) – the test data
- **y_train** (*array*) – the train target
- **y_test** (*array*) – the test target
- **parameters** (*dict*) – parameters of the model
- **metrics** (*dict*) – metrics to evaluate the model
- **name** (*str*) – name of the model (default value is the model's creation date

```
>>> model = Model(RandomForestRegressor, X_train, X_test, y_train, y_test,
                  parameters={'n_estimator': 50}, metrics={r2_score: {},
mean_squared_error: {}})
```

api.Model.**fit** ( *self, save=True, kwargs_fit={}* )

Fit the model

**Parameters**

- **save** (*bool*) – whether to save the model or not
- **kwargs_fit** (*dict*) – arguments of the fit method of the model

api.Model.**predict** ( *self, metrics={}, save=True, write=True, kwargs_predict={}* )

Test the model

**Parameters**

- **metrics** (*dict*) – metrics to evaluate the model (if metric != {}, erase the metrics passed during initialization)
- **save** (*bool*) – whether to save the model or not
- **write** (*bool*) – whether to write information in text file or not
- **kwargs_predict** (*dict*) – arguments of the predict method of the model

```
>>> model.predict(metrics={accuracy_score: {}, recall_score: {'average':
'macro'}}) # classification metrics
```

api.Model.**process** ( *self*, *metrics={}*, *save=True*, *write=True*, *kwargs_fit={}*, *kwargs_predict={}* )
Compute fit and predict

**Parameters**
- **metrics** (*dict*) – metrics to evaluate the model
- **save** (*bool*) – whether to save the model or not
- **write** (*bool*) – whether to write information in text file or not
- **kwargs_fit** (*dict*) – arguments of the fit method of the model
- **kwargs_predict** (*dict*) – arguments of the predict method of the model

api.Model.**inference** ( *self*, *X_pred=None*, *file=''* )
Run inference from the model Use either an array or a csv file

**Parameters**
- **X_pred** (*array*) – the input data
- **file** (*str*) – data file name

api.Model.**optimize** ( *self*, *parameters_range*, *metric={}*, *n_lhs=1*, *n_calls=1*, *min_or_max='min'*, *refit=False*, *verbose=True*, *write=True*, *plot_cvg=False* ) → tuple
Run Bayesian optimization (EGO) to find the best hyperparameters of the model

**Parameters**
- **parameters_range** (*dict*) – parameters with range
- **metric** (*dict*) – metric to optimize
- **n_lhs** (*int*) – numbers of initial points. Use LHS algorithm to find them
- **n_calls** (*int*) – numbers of calls to objective function. n_calls-n_lhs is the number of points computed by Bayesian optimization
- **refit** (*bool*) – whether to refit the model with the best parameters found or not
- **min_or_max** (*str*) – whether to find the minimum or maximum of the objective function
- **verbose** (*bool*) – whether to print information during optimization or not (advised for long optimization runs
- **write** (*bool*) – whether to write information in text file or not
- **plot_cvg** (*bool*) – whether to plot the convergence plot or not

**Returns** best parameters, best score, duration of optimization

**Return type** tuple

```
>>> model.optimize(parameters_range={'n_estimators': [10, 20, 30, 40],
'min_samples_split': [2,4,6,8,10]},
            n_lhs=10, n_calls=20, metric={r2_score: {}}, min_or_max='max',
refit=True)
```

## 1.2 Some functions to deal with data and model

api.**getData** ( *inputs: list*, *output: str*, *df_or_file*, *test_size=0.0*, *random_state=None* )
Get data from either file or array

**Parameters**
- **inputs** (*list*) – inputs name
- **output** (*str*) – target name
- **df_or_file** – array or csv file from which get the data
- **test_size** (*float*) – percentage of data used to test the model. Use test_size=0.0 to not split the data into train/test
- **random_state** – seed

**Returns** X, y if test_size=0.0 else X_train, X_test, y_train, y_test

**Return type** tuple of array

```
>>> X_train, X_test, y_train, y_test = getData(inputs=['a', 'b'], output='c',
df_or_file=df, test_size=0.30)
```

```
>>> X, y = getData(inputs=['a', 'b'], output='c', df_or_file='data.txt',
test_size=0.0)
```

api.**getModel** ( *file_name: str* ) → api.Model

Get the model from pickle file

**Parameters** **file_name** (*str*) – pickle file name

**Returns** The model

**Return type** Model

api.**prediction** ( *model_name: str, file_pred='', X_pred=None* )

Run inference from either array or csv file

**Parameters**
- **model_name** (*str*) – pickle file name
- **file_pred** (*str*) – data file name
- **X_pred** (*array*) – input data

```
>>> prediction(model_name=model.name, X_pred=X_pred)
```

```
>>> prediction(model_name=model.name, file_pred='data.txt')
```

- genindex
- modindex
- search

## a