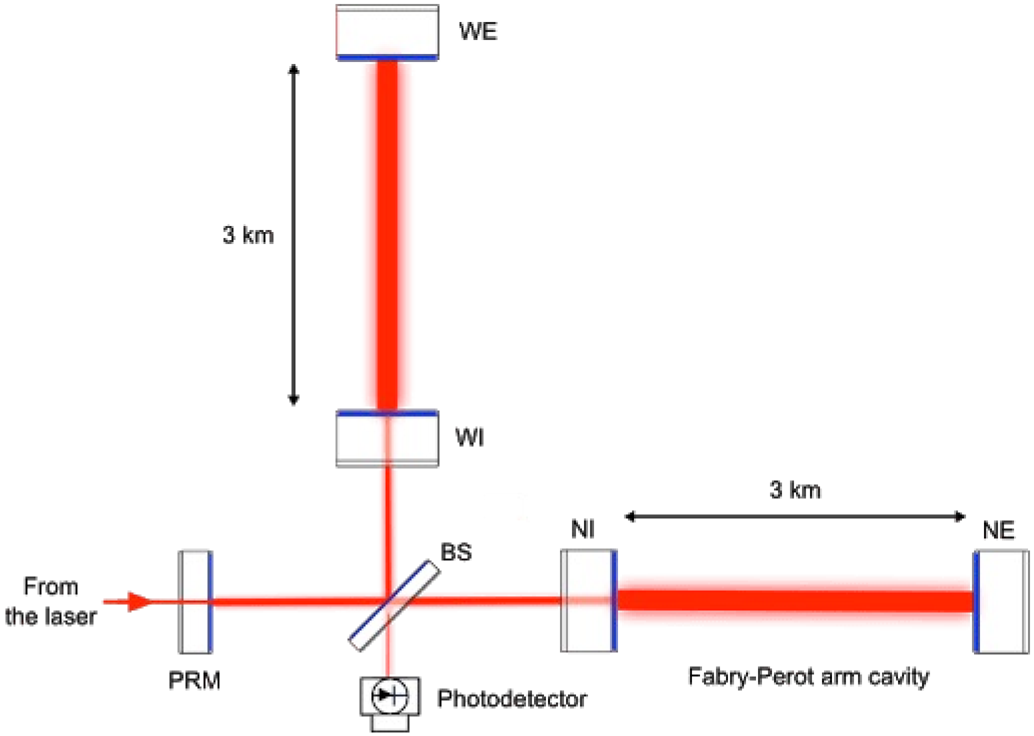
Machine learning in practice

Presentation of a simple ML task example

Table of contents

1) Context of the example
2) Tools used
3) Practice
    1) Preprocessing data
    2) Training
    3) Evaluate the algorithm

Context



Virgo interferometer

Gravitational waves

↓

Change of the relative length of the arms

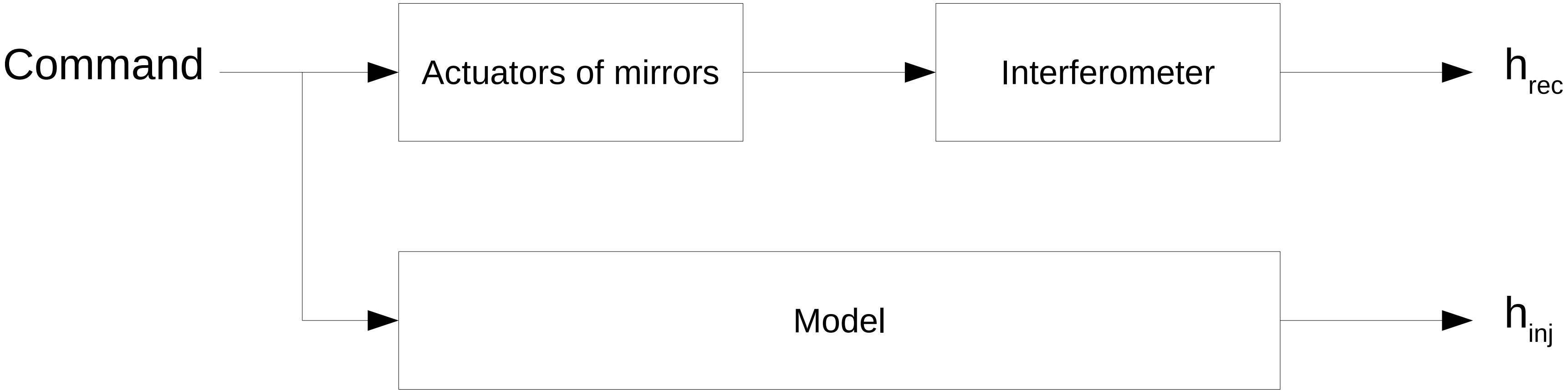
↓

Change of the interferences at the photodetector

↓

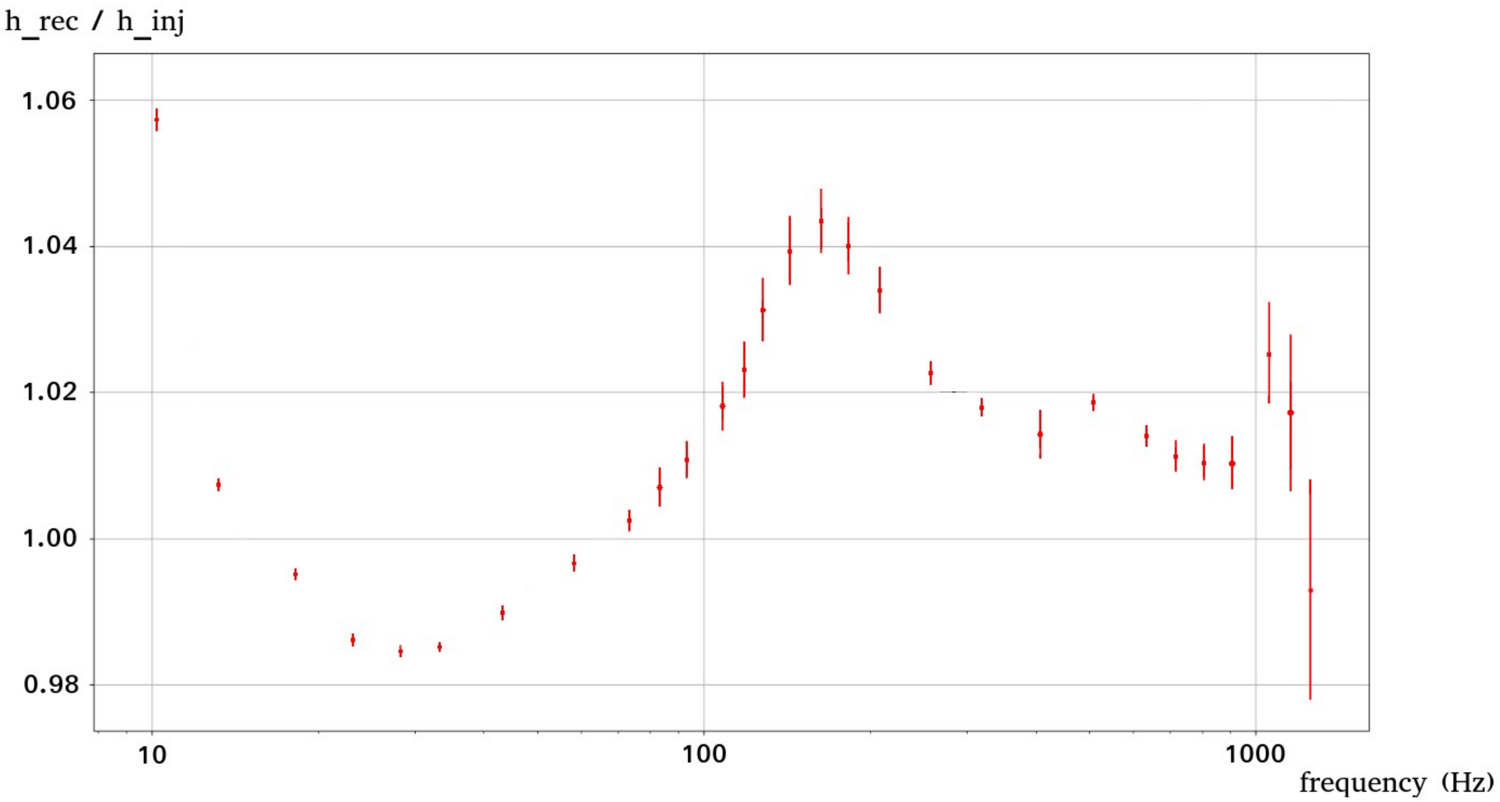
Reconstruction of gravitational waves signal

Context



We “inject” a sinusoid command at a precise frequency f, then we get the interferometer response at this frequency

Context



Context

We call:

$X = \log(f)$ the “feature”

$y = h_{rec} / h_{inj}$ the “label”

We search a function F such as:

$y = f(X)$

Tools

python™

scikit learn

Going further...

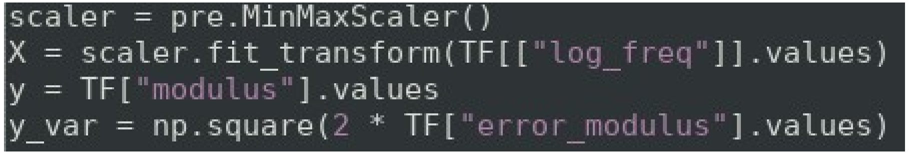
TensorFlow

PyTorch

Keras

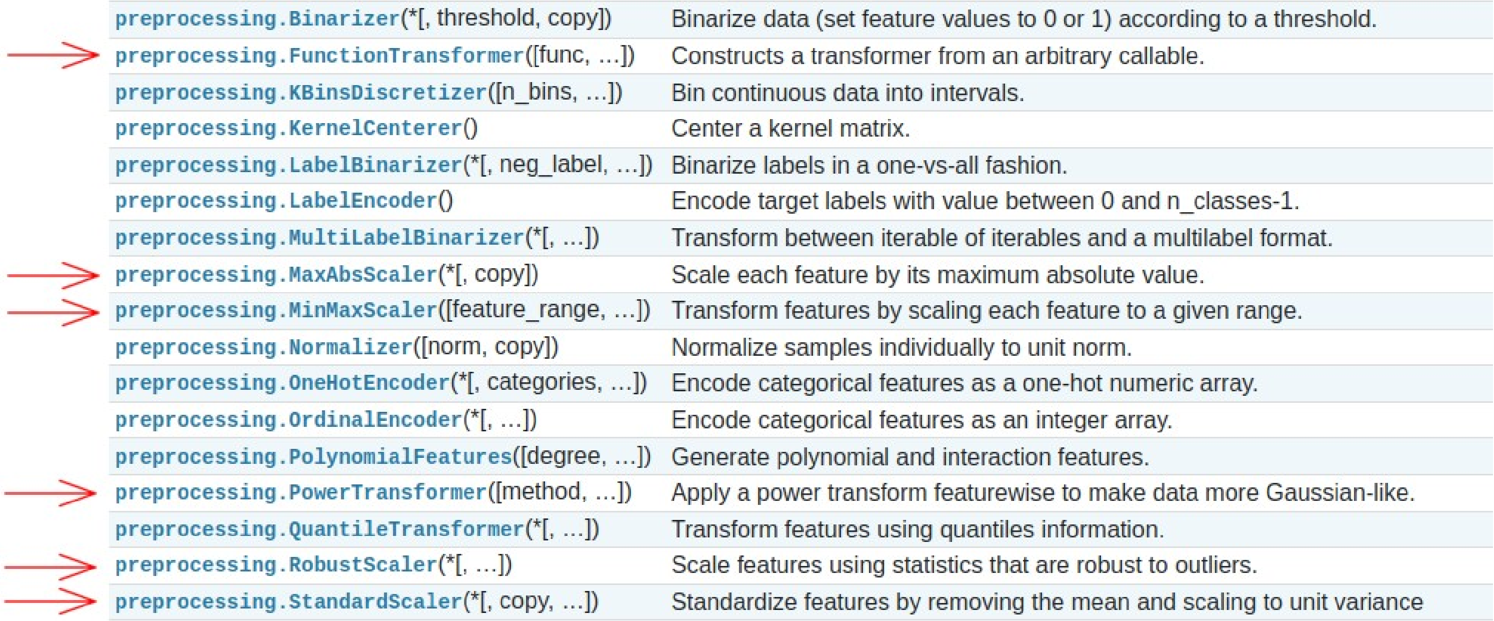
Preprocessing

Standardization :



```
scaler = pre.MinMaxScaler()
X = scaler.fit_transform(TF[["log_freq"]].values)
y = TF["modulus"].values
y_var = np.square(2 * TF["error_modulus"].values)
```

Rescale data, “MinMaxScaler” is a linear transformation which make data fit between 0 and 1



	`preprocessing.Binarizer`(*[, threshold, copy])	Binarize data (set feature values to 0 or 1) according to a threshold.
→	`preprocessing.FunctionTransformer`([func, …])	Constructs a transformer from an arbitrary callable.
	`preprocessing.KBinsDiscretizer`([n_bins, …])	Bin continuous data into intervals.
	`preprocessing.KernelCenterer`()	Center a kernel matrix.
	`preprocessing.LabelBinarizer`(*[, neg_label, …])	Binarize labels in a one-vs-all fashion.
	`preprocessing.LabelEncoder`()	Encode target labels with value between 0 and n_classes-1.
	`preprocessing.MultiLabelBinarizer`(*[, …])	Transform between iterable of iterables and a multilabel format.
→	`preprocessing.MaxAbsScaler`(*[, copy])	Scale each feature by its maximum absolute value.
→	`preprocessing.MinMaxScaler`([feature_range, …])	Transform features by scaling each feature to a given range.
	`preprocessing.Normalizer`([norm, copy])	Normalize samples individually to unit norm.
	`preprocessing.OneHotEncoder`(*[, categories, …])	Encode categorical features as a one-hot numeric array.
	`preprocessing.OrdinalEncoder`(*[, …])	Encode categorical features as an integer array.
	`preprocessing.PolynomialFeatures`([degree, …])	Generate polynomial and interaction features.
→	`preprocessing.PowerTransformer`([method, …])	Apply a power transform featurewise to make data more Gaussian-like.
	`preprocessing.QuantileTransformer`(*[, …])	Transform features using quantiles information.
→	`preprocessing.RobustScaler`(*[, …])	Scale features using statistics that are robust to outliers.
→	`preprocessing.StandardScaler`(*[, copy, …])	Standardize features by removing the mean and scaling to unit variance

Gaussian process regression

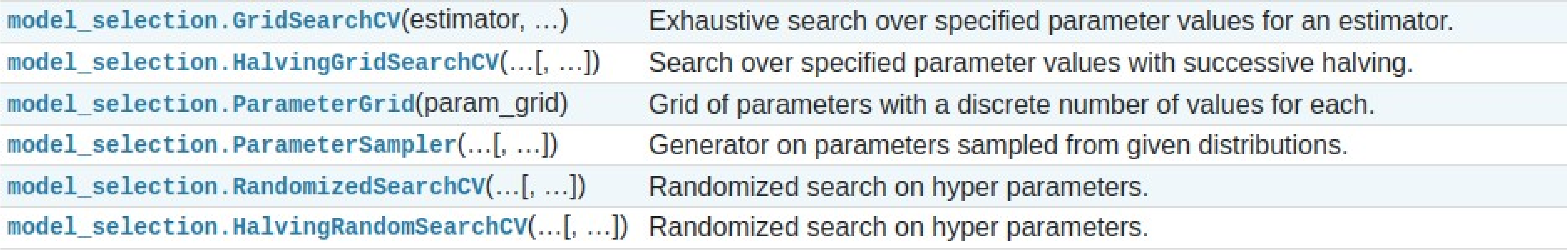
- A gaussian process is a set of gaussian random variables, indexed by a variable x,
- It is characterized by its mean “µ(x)”, and its covariance “cov(x_1,x_2)”

The covariance used in this example: $cov(x_1, x_2) = \gamma_1^2 + \gamma_2^2 \times \exp\left(\frac{(x_1 - x_2)^2}{2l^2}\right)$

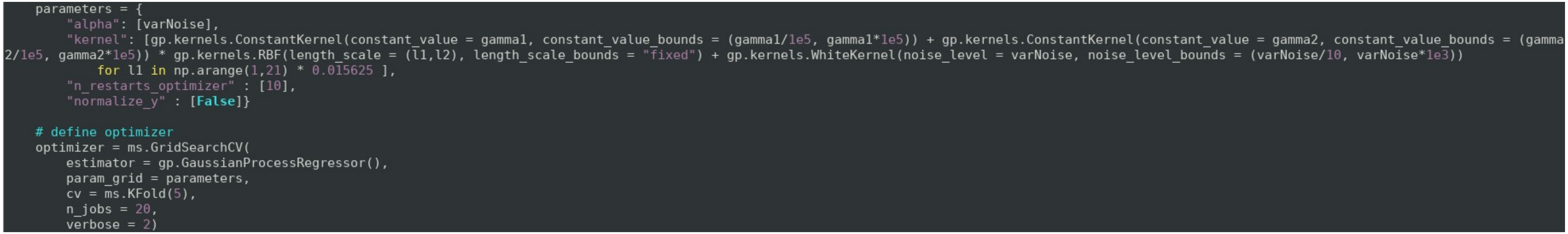
The regression is computed thanks to the input data, and the covariance matrix

Tuning hyperparameters

Make a set of hyperparameter values, and test the algorithm with each hyperparameter value.



`model_selection.GridSearchCV`(estimator, …)	Exhaustive search over specified parameter values for an estimator.
`model_selection.HalvingGridSearchCV`(…[, …])	Search over specified parameter values with successive halving.
`model_selection.ParameterGrid`(param_grid)	Grid of parameters with a discrete number of values for each.
`model_selection.ParameterSampler`(…[, …])	Generator on parameters sampled from given distributions.
`model_selection.RandomizedSearchCV`(…[, …])	Randomized search on hyper parameters.
`model_selection.HalvingRandomSearchCV`(…[, …])	Randomized search on hyper parameters.

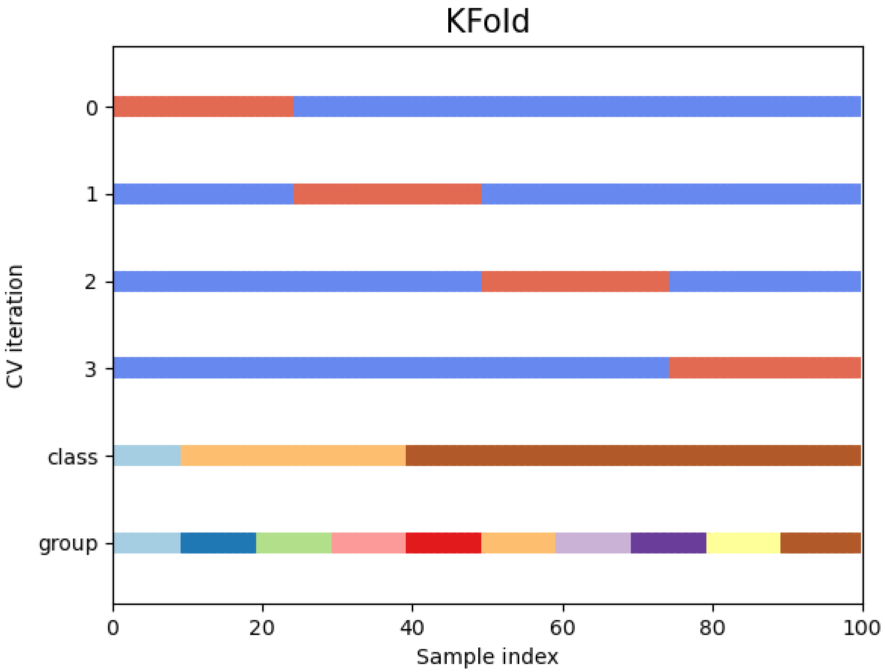


```
    parameters = {
        "alpha": [varNoise],
        "kernel": [gp.kernels.ConstantKernel(constant_value = gamma1, constant_value_bounds = (gamma1/1e5, gamma1*1e5)) + gp.kernels.ConstantKernel(constant_value = gamma2, constant_value_bounds = (gamma
2/1e5, gamma2*1e5)) * gp.kernels.RBF(length_scale = (l1,l2), length_scale_bounds = "fixed") + gp.kernels.WhiteKernel(noise_level = varNoise, noise_level_bounds = (varNoise/10, varNoise*1e3))
            for l1 in np.arange(1,21) * 0.015625 ],
        "n_restarts_optimizer" : [10],
        "normalize_y" : [False]}

    # define optimizer
    optimizer = ms.GridSearchCV(
        estimator = gp.GaussianProcessRegressor(),
        param_grid = parameters,
        cv = ms.KFold(5),
        n_jobs = 20,
        verbose = 2)
```

Evaluate an algorithm

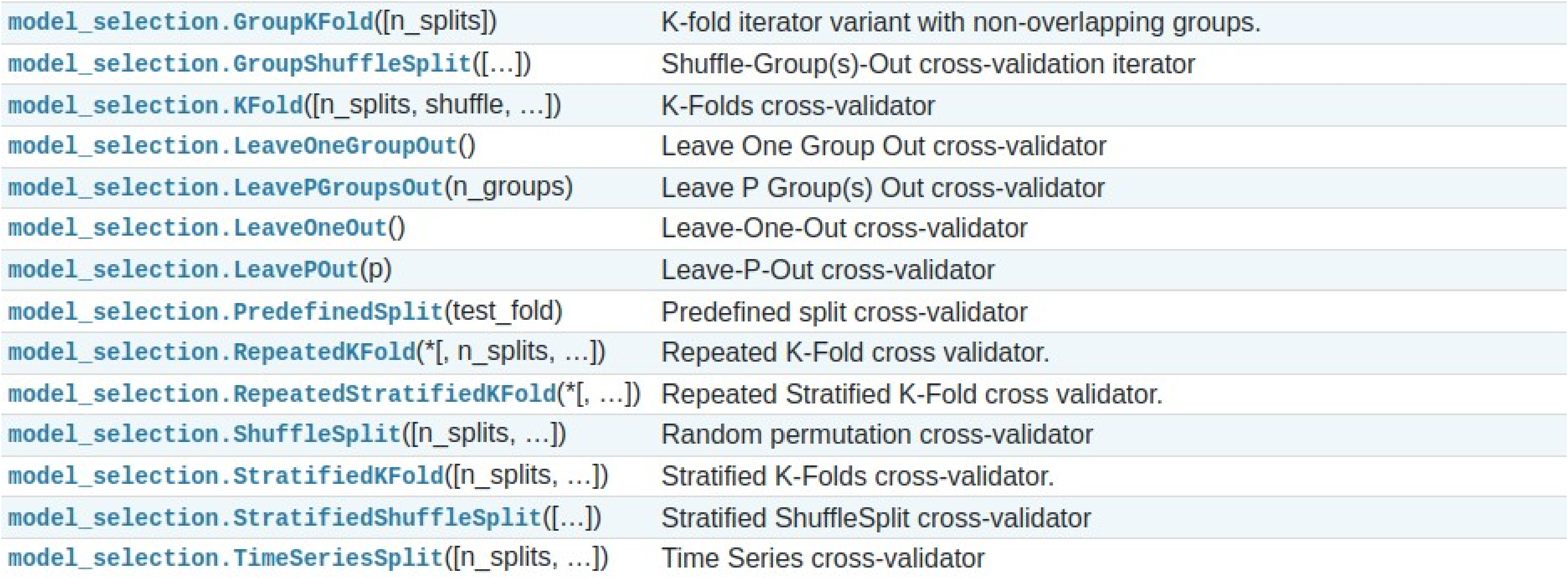
Cross validation



- Split the dataset into two: The trainng dataset and the testing dataset,
- Train the algorithm with the Training dataset
- Predict the labels of the testing dataset,
- Compare the predicted labels with the true value of the label

Evaluate an algorithm

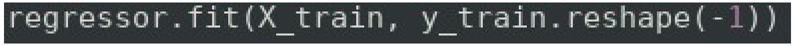
There are many algorithms to split the dataset,



model_selection.GroupKFold([n_splits])	K-fold iterator variant with non-overlapping groups.
model_selection.GroupShuffleSplit([…])	Shuffle-Group(s)-Out cross-validation iterator
model_selection.KFold([n_splits, shuffle, …])	K-Folds cross-validator
model_selection.LeaveOneGroupOut()	Leave One Group Out cross-validator
model_selection.LeavePGroupsOut(n_groups)	Leave P Group(s) Out cross-validator
model_selection.LeaveOneOut()	Leave-One-Out cross-validator
model_selection.LeavePOut(p)	Leave-P-Out cross-validator
model_selection.PredefinedSplit(test_fold)	Predefined split cross-validator
model_selection.RepeatedKFold(*[, n_splits, …])	Repeated K-Fold cross validator.
model_selection.RepeatedStratifiedKFold(*[, …])	Repeated Stratified K-Fold cross validator.
model_selection.ShuffleSplit([n_splits, …])	Random permutation cross-validator
model_selection.StratifiedKFold([n_splits, …])	Stratified K-Folds cross-validator.
model_selection.StratifiedShuffleSplit([…])	Stratified ShuffleSplit cross-validator
model_selection.TimeSeriesSplit([n_splits, …])	Time Series cross-validator

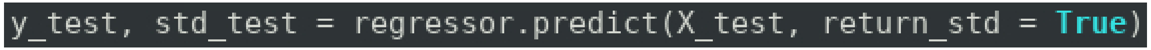
Make prediction

- Define a regressor with optimized hyperparameters and fit model with data



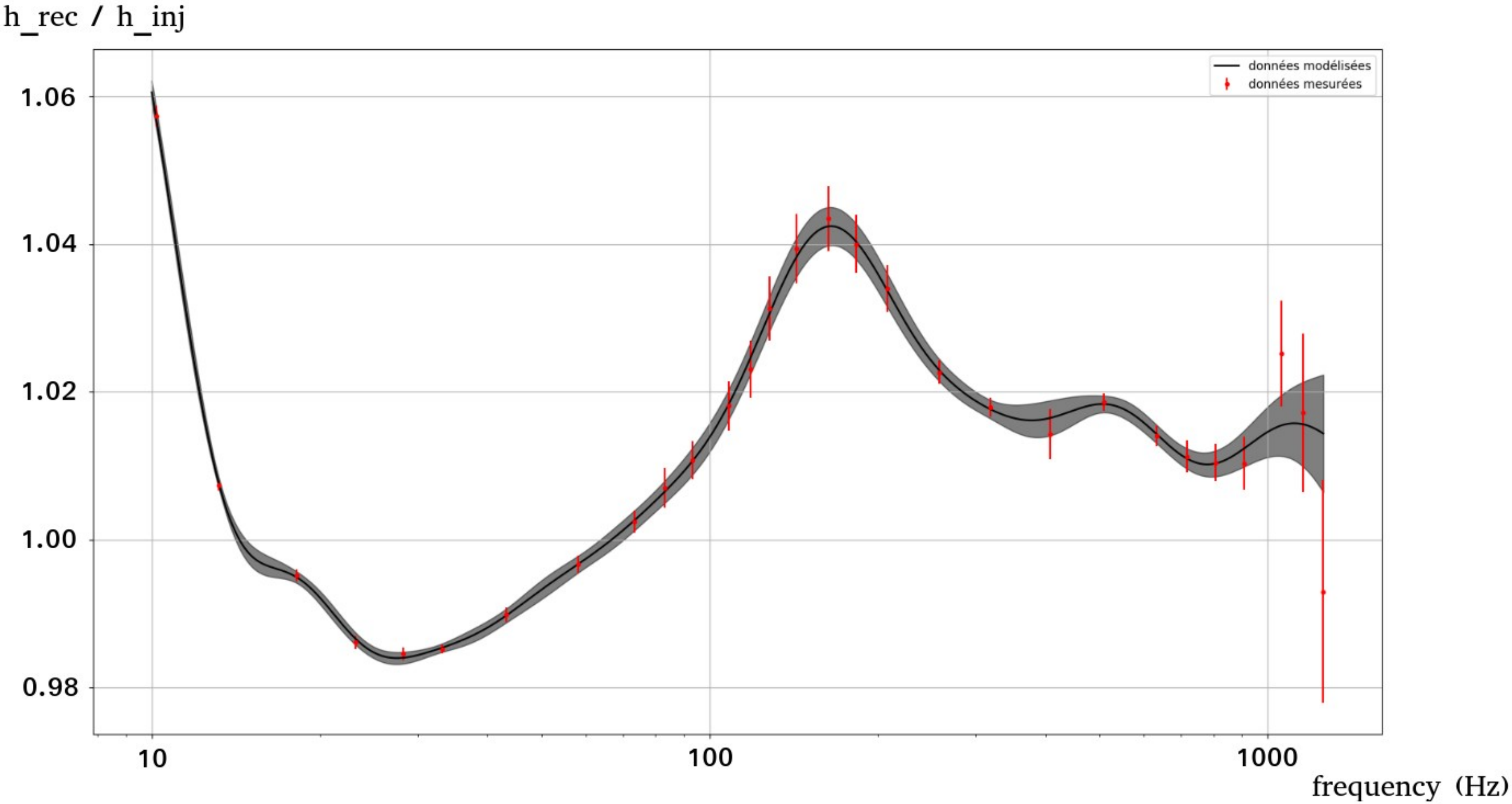
```
regressor.fit(X_train, y_train.reshape(-1))
```

- Predict values of label at the test dataset



```
y_test, std_test = regressor.predict(X_test, return_std = True)
```

Make prediction



Thank you for your attention