SCIKIT-LEARN TUTORIAL: MACHINE LEARNING IN PYTHON

MENGYING SUN, **TRANSLATIONAL BIOINFORMATICS ONLINE WORKSHOP** 08/12

OUTLINE

What is Scikit-learn?

Machine learning with scikit-learn

- Data preprocessing
- Build the pipeline (modeling)
- Cross validation (grid search)

Examples

- Random Forest with scikit-learn
- XGBoost model with scikit-learn
- Other examples: clustering, dimension reduction, lasso

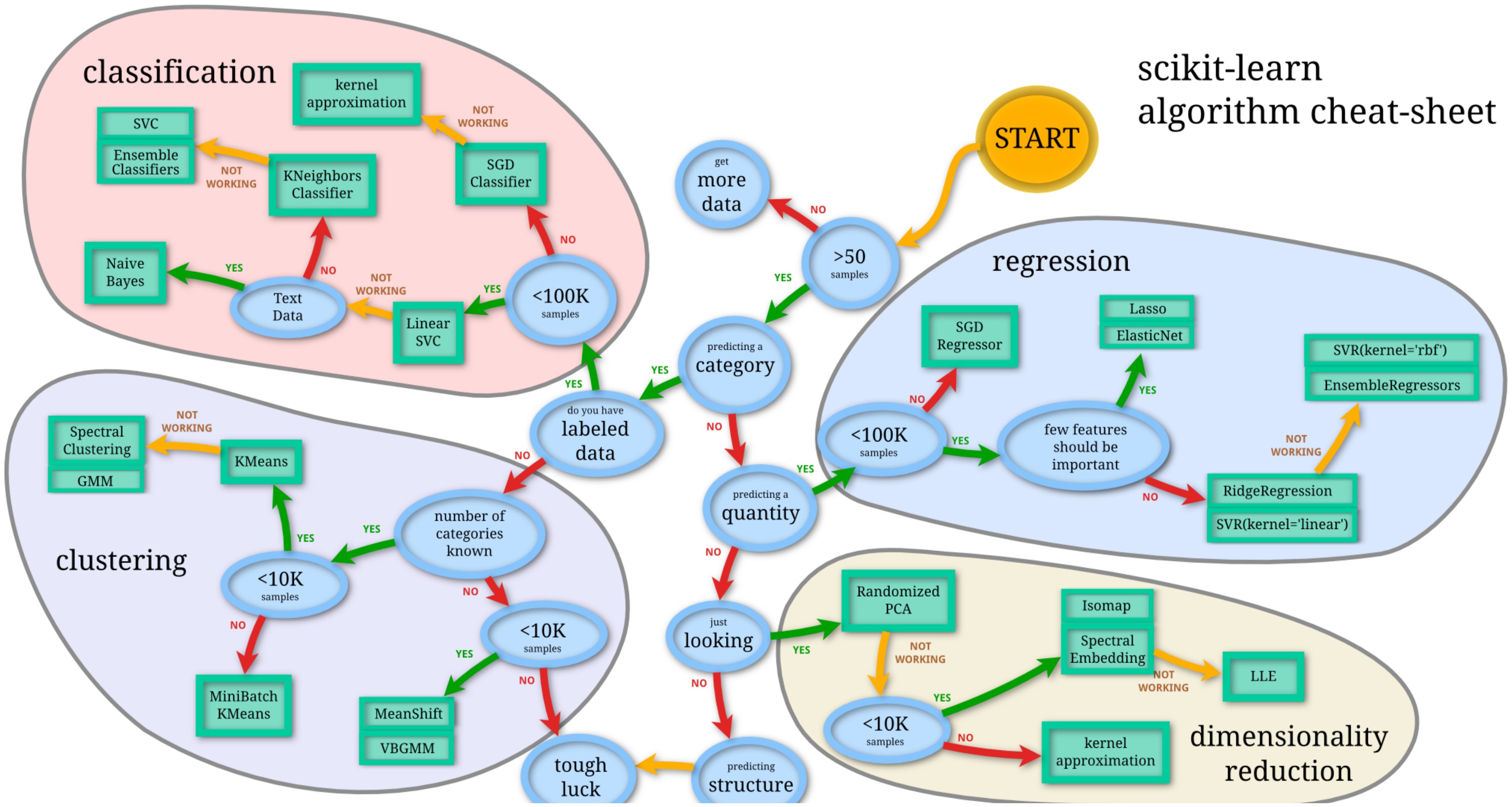
WHAT IS SCIKIT-LEARN

KEY FEATURE

WHAT IS SCIKIT-LEARN

- An open source Python library for machine learning
  - Preprocessing
  - Dimensionality reduction
  - Classification / regression / clustering
  - Model selection
  - Deep Learning (MLP)

WHAT IS SCIKIT-LEARN



Map Link

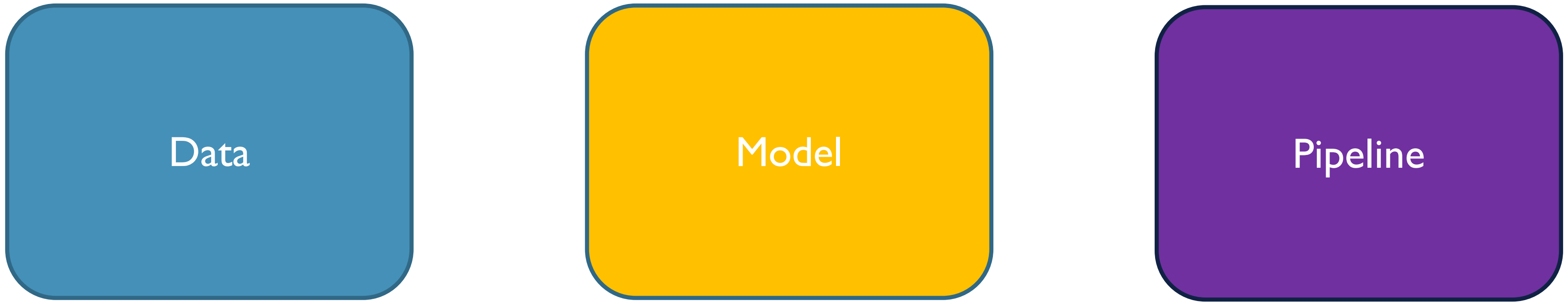
KEY FEATURE OF SCIKIT-LEARN

- **Highly modularized**
  - Same coding structures for different
    - Algorithms (SVM, Random Forest, Boosting, MLP etc.)
    - Data pre-processing that belongs to the same category (Data splitting)

MACHINE LEARNING WITH SCIKIT-LEARN

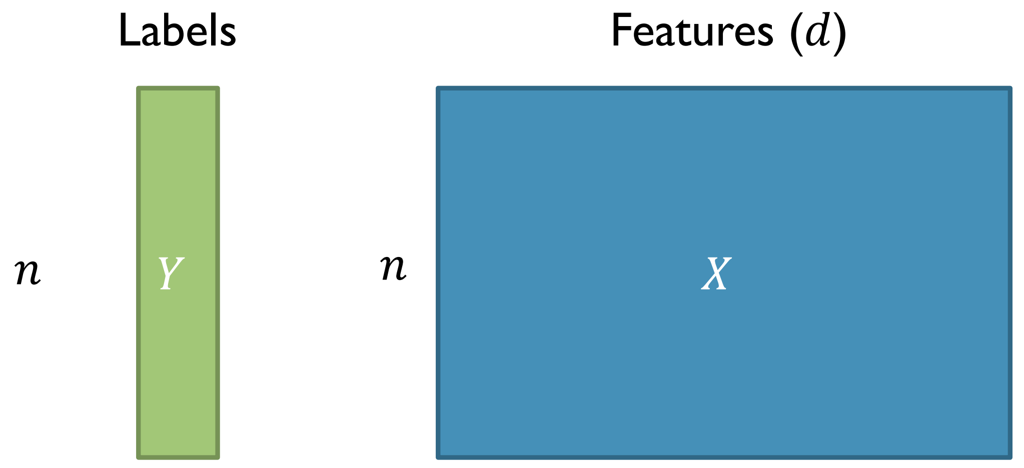
SUPERVISED LEARNING / DATA PREPROCESSING / MODEL PIPELINE / PARAMETER TUNING

SUPERVISED LEARNING



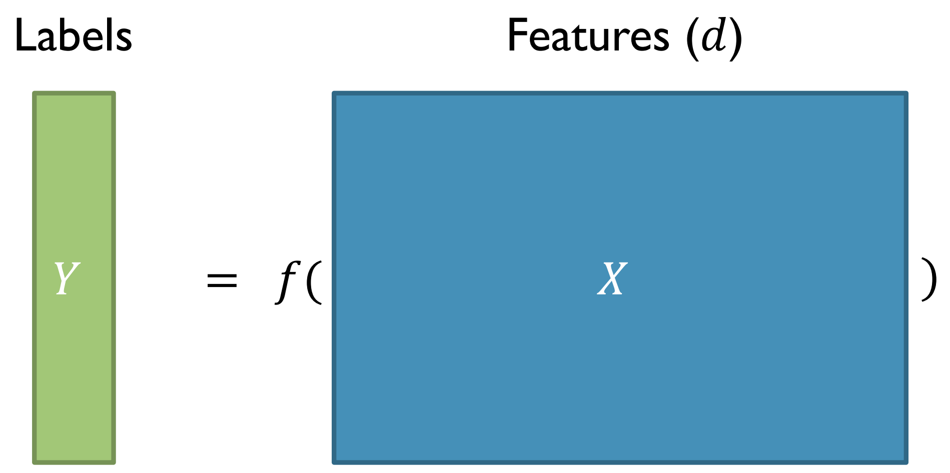
SUPERVISED LEARNING

- Features (independent variables / predictors)
  - $n \times d$
- Labels (dependent variable / target)
  - $n \times 1$



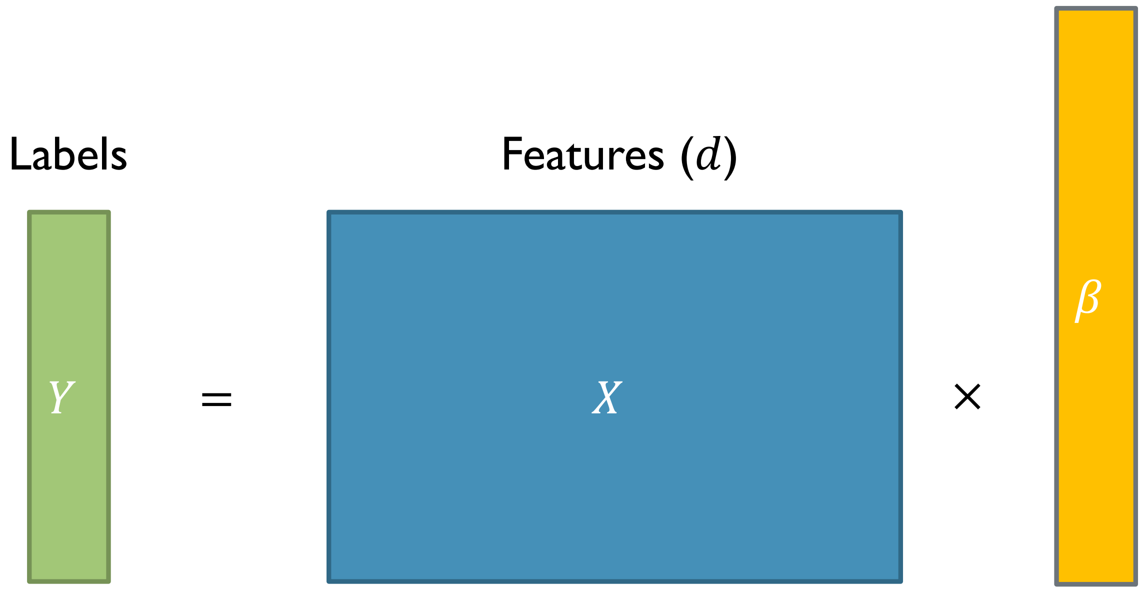
SUPERVISED LEARNING

- Features (independent variables / predictors)
  - $n \times d$
- Labels (dependent variable / target)
  - $n \times 1$



SUPERVISED LEARNING

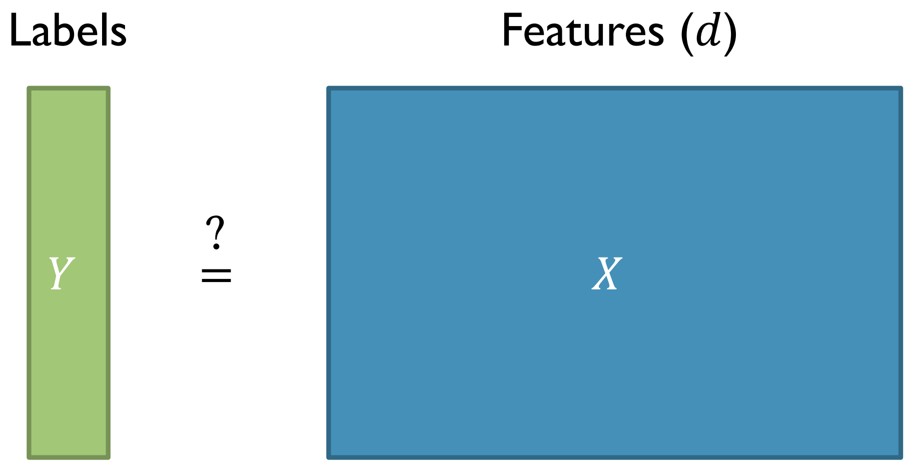
- Features (independent variables / predictors)
  - $n \times d$
- Labels (dependent variable / target)
  - $n \times 1$



Linear Regression

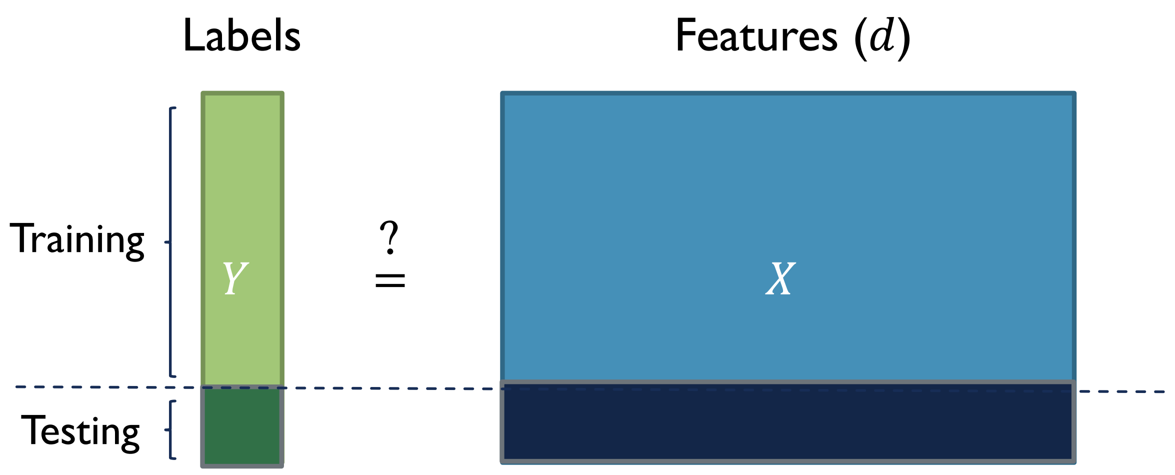
SUPERVISED LEARNING

- Statistical Analysis
  - Linear relationship
  - Focus: significance of variables
  - Simple models
- Supervised Learning
  - Nonlinear relationship
  - Focus: prediction accuracy
  - **Complex models**



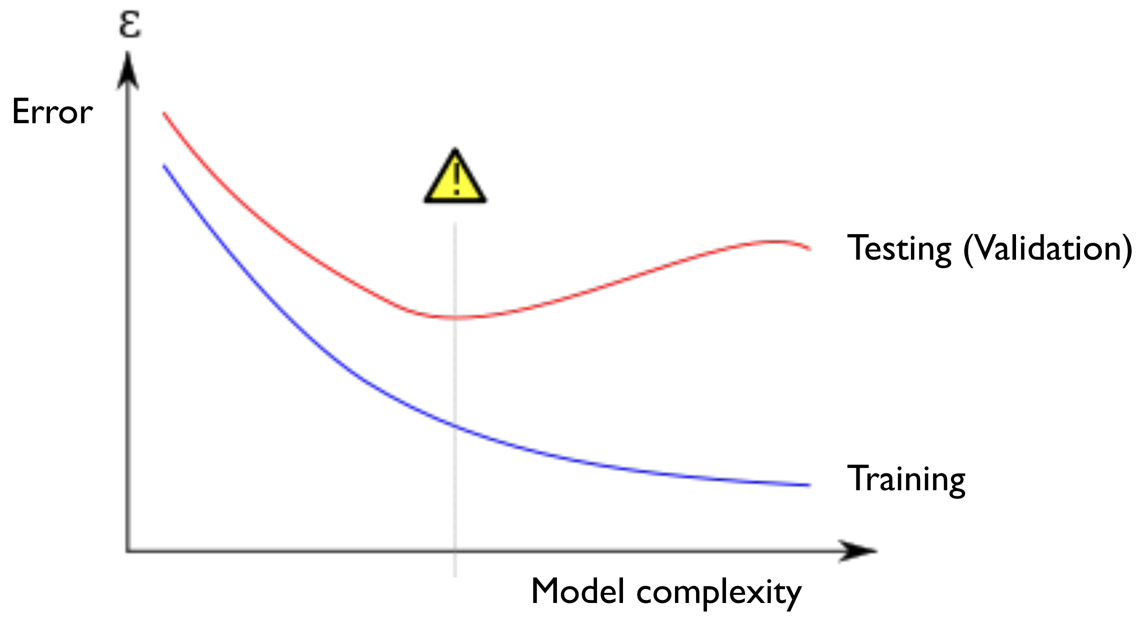
OVERFITTING

- Complex models leads to overfitting
  - Training vs Testing (Validation) data
  - Perfectly fit (memorize) data
  - However, this model is not generalizable



OVERFITTING

- Complex models leads to overfitting
  - Training vs Testing (Validation) data
  - Perfectly fit (memorize) data
  - However, this model is not generalizable

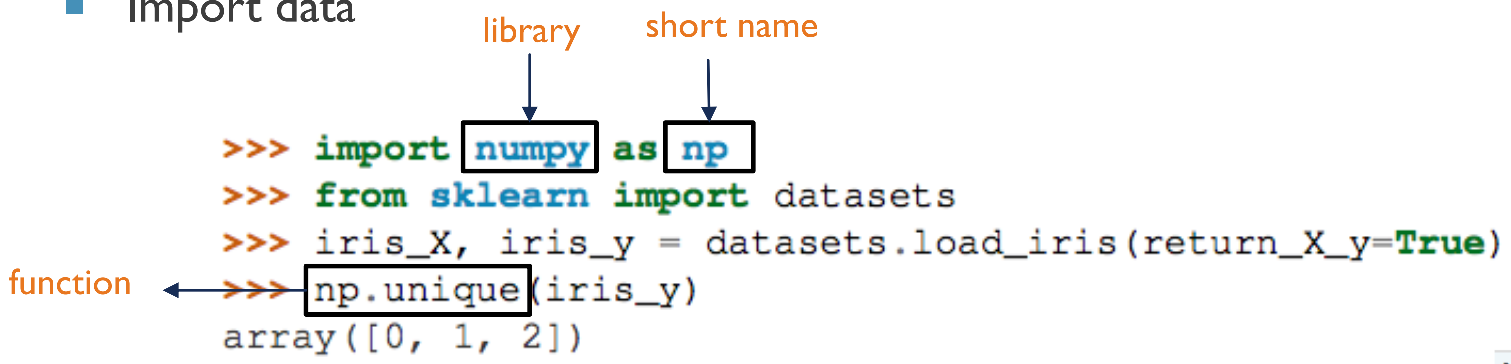


OVERALL PIPELINE

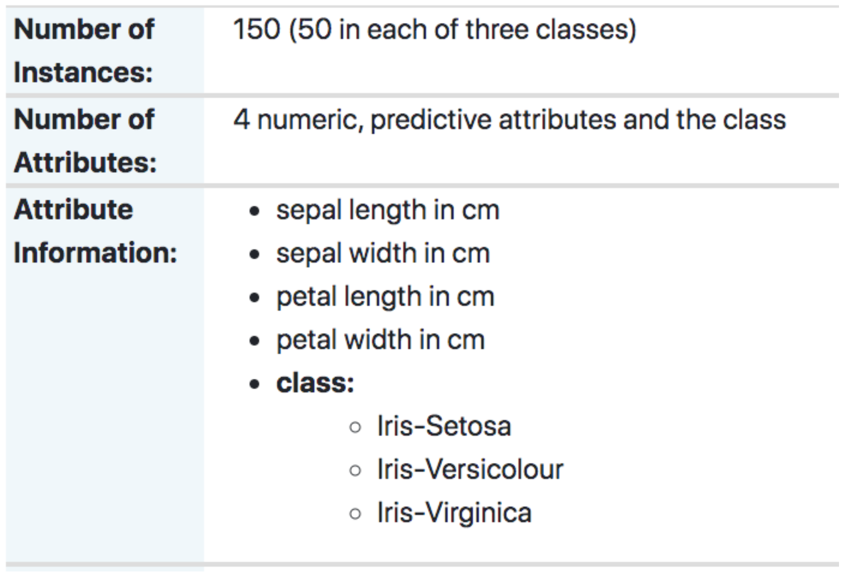
- Processed data
- Model to use
  - Overfitting
  - Hyper-parameters
- **Cross validation**
  - To get the best model

DATA PREPROCESSING

- Import data



```
>>> import numpy as np
>>> from sklearn import datasets
>>> iris_X, iris_y = datasets.load_iris(return_X_y=True)
>>> np.unique(iris_y)
array([0, 1, 2])
```

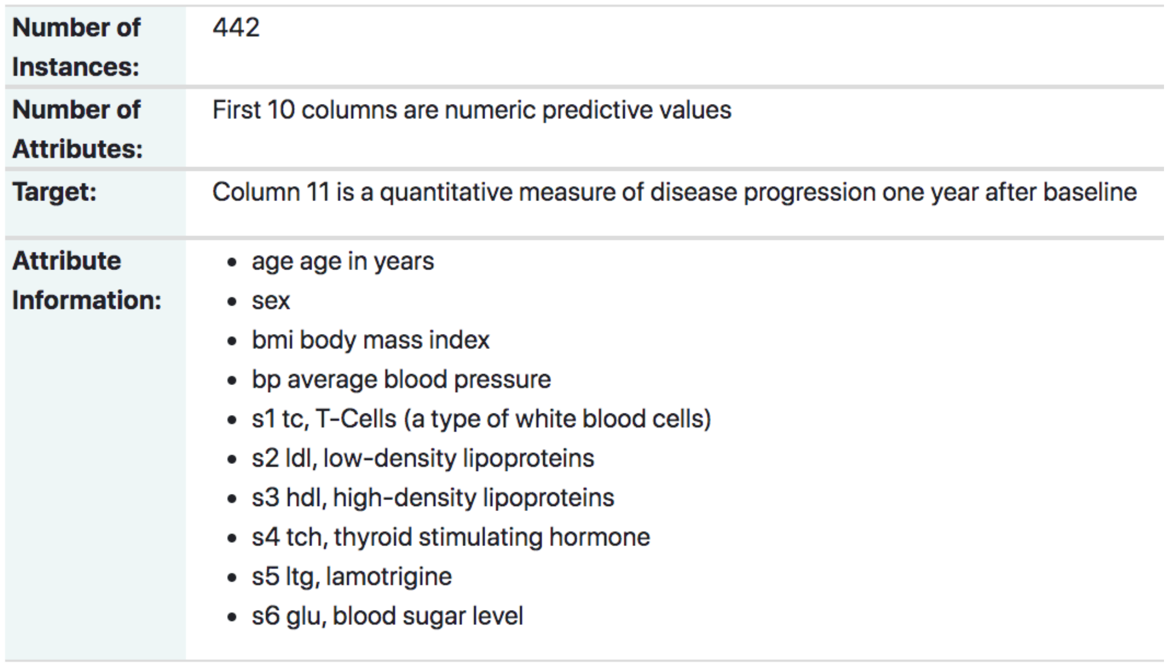


Number of Instances:	150 (50 in each of three classes)
Number of Attributes:	4 numeric, predictive attributes and the class
Attribute Information:	• sepal length in cm • sepal width in cm • petal length in cm • petal width in cm • **class:** ◦ Iris-Setosa ◦ Iris-Versicolour ◦ Iris-Virginica

DATA PREPROCESSING

- Import data

```
>>> diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)
```

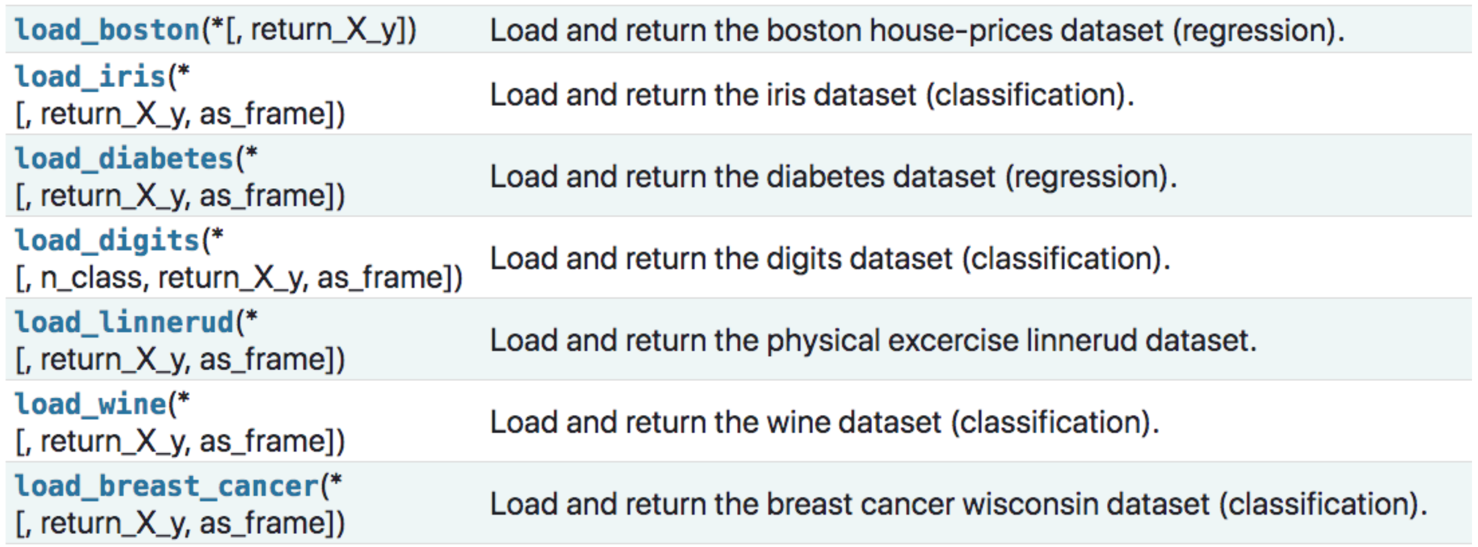


Number of Instances:	442
Number of Attributes:	First 10 columns are numeric predictive values
Target:	Column 11 is a quantitative measure of disease progression one year after baseline
Attribute Information:	• age age in years • sex • bmi body mass index • bp average blood pressure • s1 tc, T-Cells (a type of white blood cells) • s2 ldl, low-density lipoproteins • s3 hdl, high-density lipoproteins • s4 tch, thyroid stimulating hormone • s5 ltg, lamotrigine • s6 glu, blood sugar level

DATA PREPROCESSING

- Import data

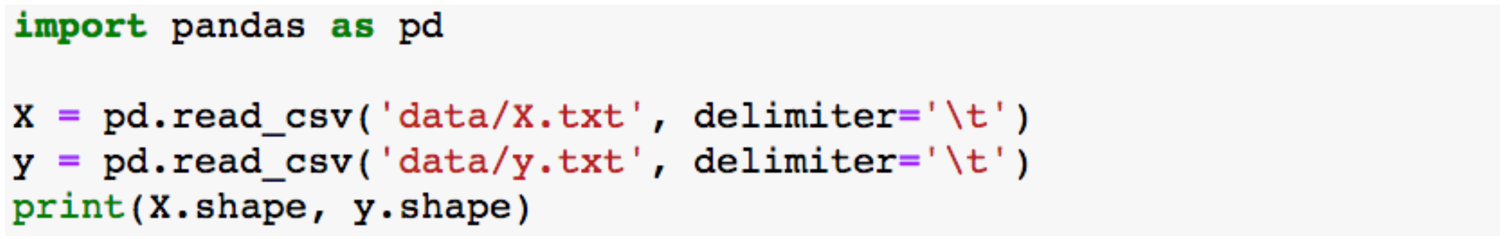
```
>>> diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)
```



`load_boston`(*[, return_X_y])	Load and return the boston house-prices dataset (regression).
`load_iris`(* [, return_X_y, as_frame])	Load and return the iris dataset (classification).
`load_diabetes`(* [, return_X_y, as_frame])	Load and return the diabetes dataset (regression).
`load_digits`(* [, n_class, return_X_y, as_frame])	Load and return the digits dataset (classification).
`load_linnerud`(* [, return_X_y, as_frame])	Load and return the physical excercise linnerud dataset.
`load_wine`(* [, return_X_y, as_frame])	Load and return the wine dataset (classification).
`load_breast_cancer`(* [, return_X_y, as_frame])	Load and return the breast cancer wisconsin dataset (classification).

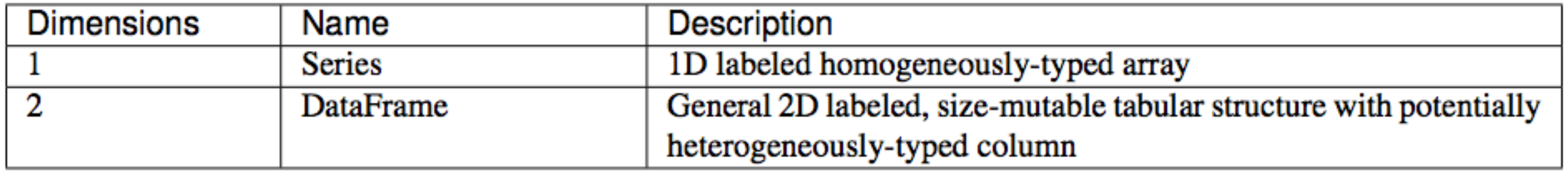
DATA PREPROCESSING

- Import user defined data
  - Pandas (Data analysis library in Python)



```
import pandas as pd

X = pd.read_csv('data/X.txt', delimiter='\t')
y = pd.read_csv('data/y.txt', delimiter='\t')
print(X.shape, y.shape)
```



Dimensions	Name	Description
1	Series	1D labeled homogeneously-typed array
2	DataFrame	General 2D labeled, size-mutable tabular structure with potentially heterogeneously-typed column

c() in r

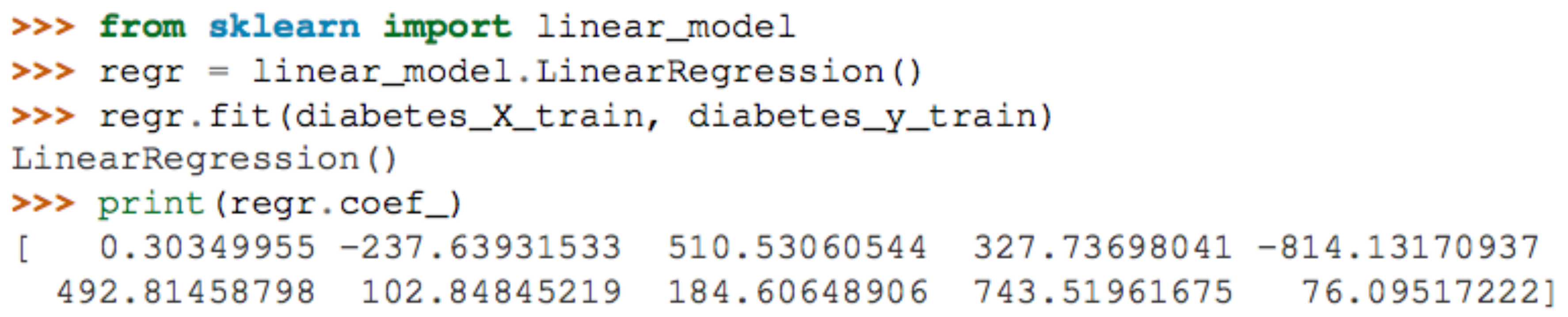
df/matrix in r

DATA PREPROCESSING

- Functionality of Pandas
  - Read/write
  - Filter/extract
  - Min, max etc.
  - Aggregation, merge, apply etc.
- You can almost always find the function corresponding to R(base)

BUILDING A MODEL

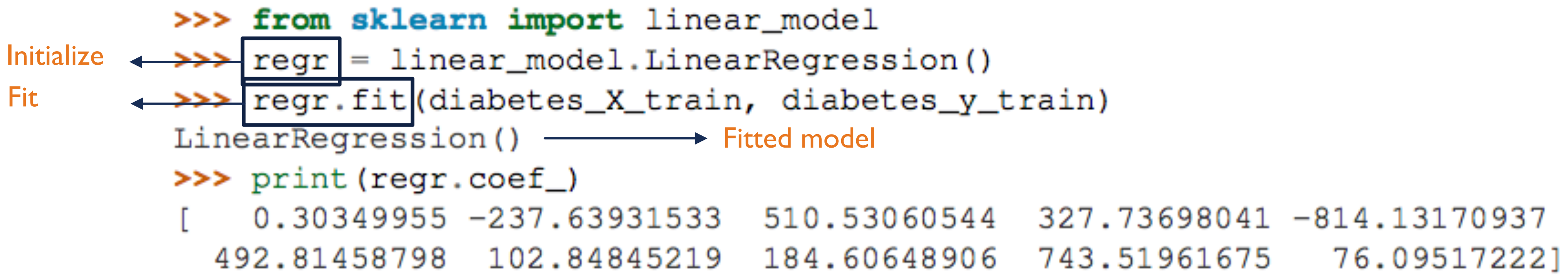
- Linear regression model
  - See Jupyter notebook



```
>>> from sklearn import linear_model
>>> regr = linear_model.LinearRegression()
>>> regr.fit(diabetes_X_train, diabetes_y_train)
LinearRegression()
>>> print(regr.coef_)
[   0.30349955 -237.63931533  510.53060544  327.73698041 -814.13170937
  492.81458798  102.84845219  184.60648906  743.51961675   76.09517222]
```

BUILDING A MODEL

- Linear regression model
  - See Jupyter notebook



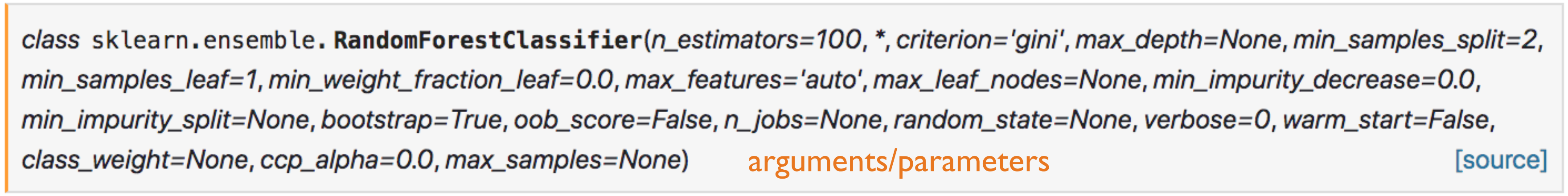
```
>>> from sklearn import linear_model
>>> regr = linear_model.LinearRegression()
>>> regr.fit(diabetes_X_train, diabetes_y_train)
LinearRegression()
>>> print(regr.coef_)
[   0.30349955 -237.63931533  510.53060544  327.73698041 -814.13170937
  492.81458798  102.84845219  184.60648906  743.51961675   76.09517222]
```

BUILDING A MODEL

- Tree Model
  - Simple Example (regression, see Jupyter)
  - Sensitivity of parameters → cross validation

MODULARIZATION

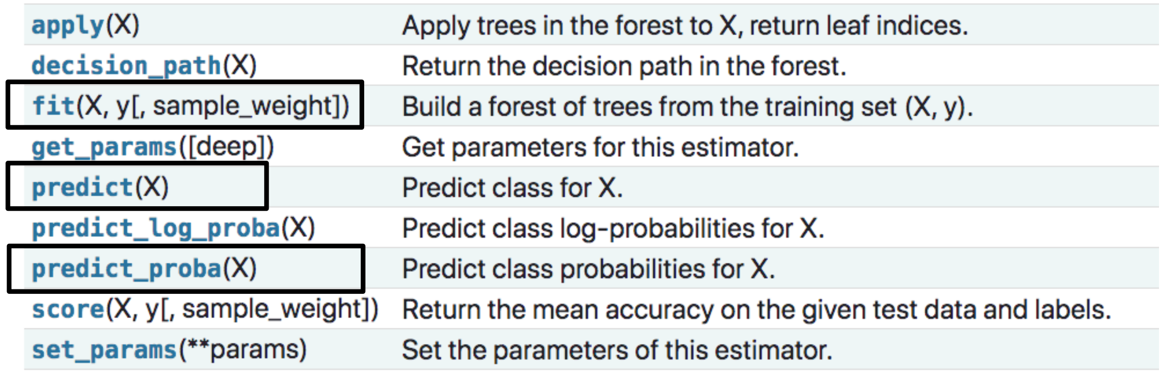
- From specific classifier to general model class (module)
  - Eg., Classification



```
class sklearn.ensemble.RandomForestClassifier(n_estimators=100, *, criterion='gini', max_depth=None, min_samples_split=2,
min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False,
class_weight=None, ccp_alpha=0.0, max_samples=None)
```

arguments/parameters [source]

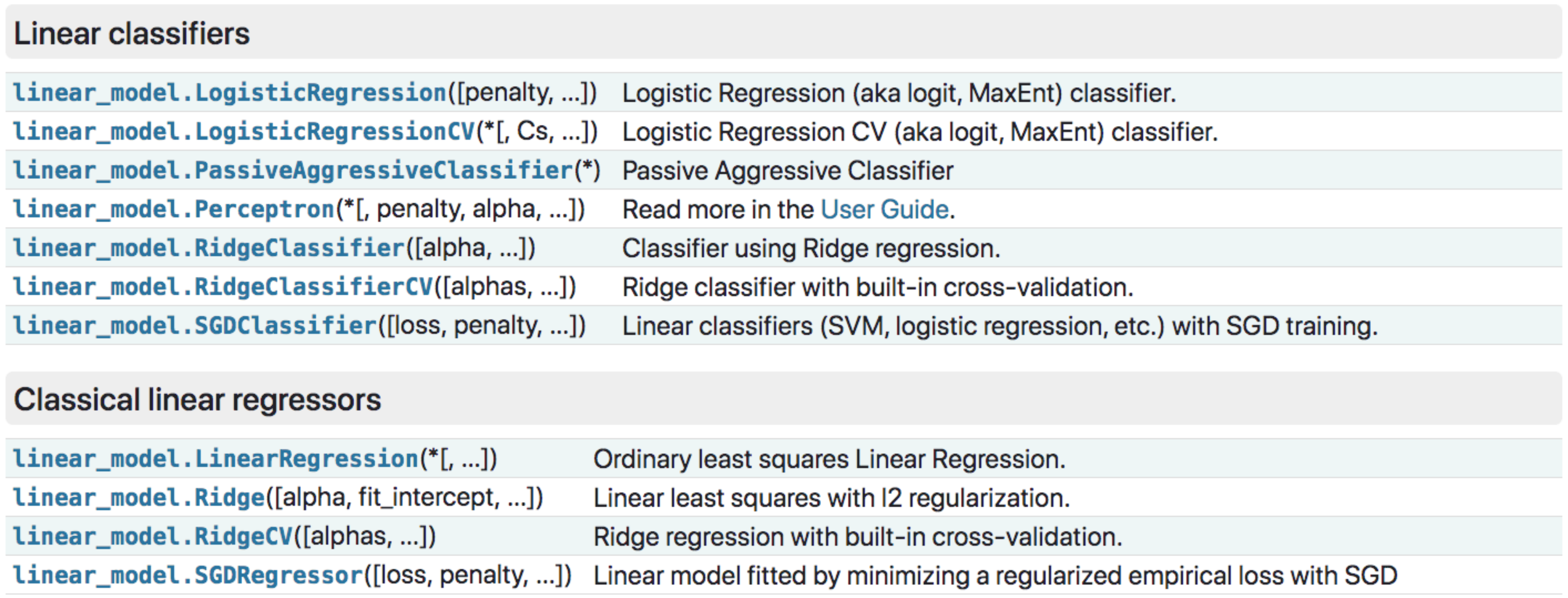
  - Methods



apply(X)	Apply trees in the forest to X, return leaf indices.
decision_path(X)	Return the decision path in the forest.
fit(X, y[, sample_weight])	Build a forest of trees from the training set (X, y).
get_params([deep])	Get parameters for this estimator.
predict(X)	Predict class for X.
predict_log_proba(X)	Predict class log-probabilities for X.
predict_proba(X)	Predict class probabilities for X.
score(X, y[, sample_weight])	Return the mean accuracy on the given test data and labels.
set_params(**params)	Set the parameters of this estimator.

MODULARIZATION

- From specific classifier to general model class (module)



Linear classifiers

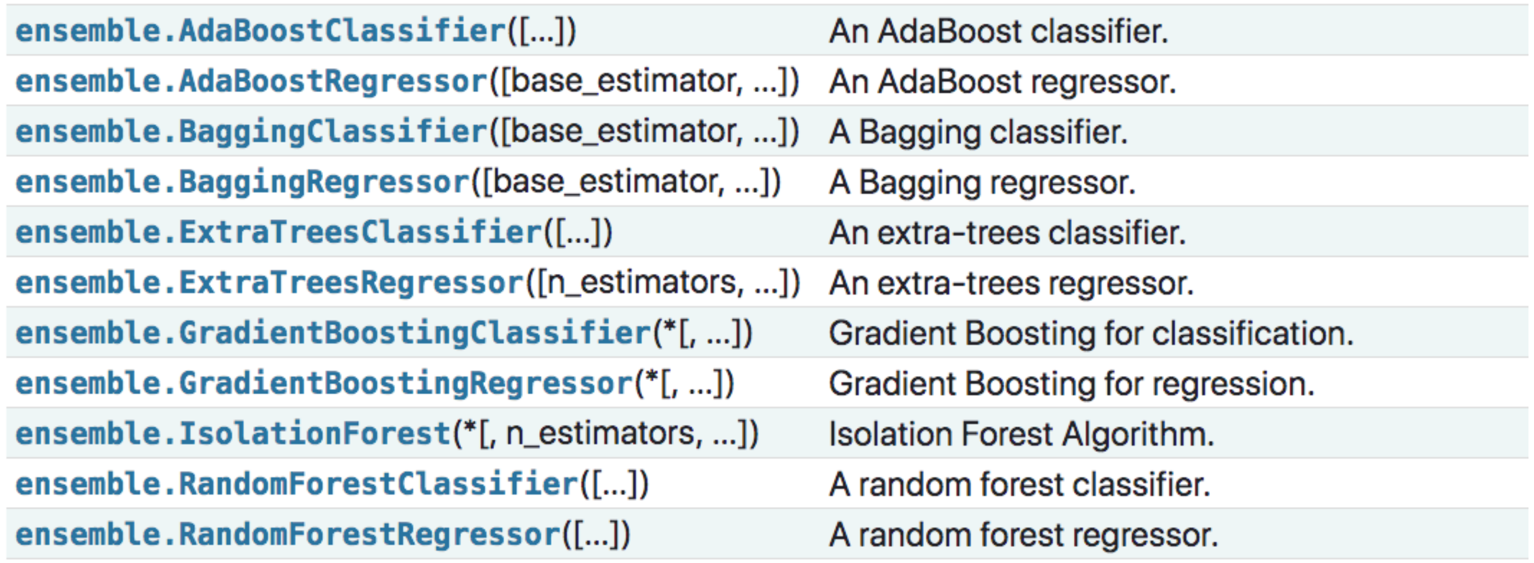
`linear_model.LogisticRegression`([penalty, …])	Logistic Regression (aka logit, MaxEnt) classifier.
`linear_model.LogisticRegressionCV`(*[, Cs, …])	Logistic Regression CV (aka logit, MaxEnt) classifier.
`linear_model.PassiveAggressiveClassifier`(*)	Passive Aggressive Classifier
`linear_model.Perceptron`(*[, penalty, alpha, …])	Read more in the User Guide.
`linear_model.RidgeClassifier`([alpha, …])	Classifier using Ridge regression.
`linear_model.RidgeClassifierCV`([alphas, …])	Ridge classifier with built-in cross-validation.
`linear_model.SGDClassifier`([loss, penalty, …])	Linear classifiers (SVM, logistic regression, etc.) with SGD training.

Classical linear regressors

`linear_model.LinearRegression`(*[, …])	Ordinary least squares Linear Regression.
`linear_model.Ridge`([alpha, fit_intercept, …])	Linear least squares with l2 regularization.
`linear_model.RidgeCV`([alphas, …])	Ridge regression with built-in cross-validation.
`linear_model.SGDRegressor`([loss, penalty, …])	Linear model fitted by minimizing a regularized empirical loss with SGD

MODULARIZATION

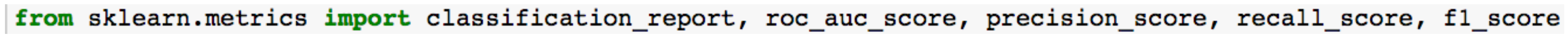
- From specific classifier to general model class (module)
  - Ensemble (non-linear) methods



`ensemble.AdaBoostClassifier`([…])	An AdaBoost classifier.
`ensemble.AdaBoostRegressor`([base_estimator, …])	An AdaBoost regressor.
`ensemble.BaggingClassifier`([base_estimator, …])	A Bagging classifier.
`ensemble.BaggingRegressor`([base_estimator, …])	A Bagging regressor.
`ensemble.ExtraTreesClassifier`([…])	An extra-trees classifier.
`ensemble.ExtraTreesRegressor`([n_estimators, …])	An extra-trees regressor.
`ensemble.GradientBoostingClassifier`(*[, …])	Gradient Boosting for classification.
`ensemble.GradientBoostingRegressor`(*[, …])	Gradient Boosting for regression.
`ensemble.IsolationForest`(*[, n_estimators, …])	Isolation Forest Algorithm.
`ensemble.RandomForestClassifier`([…])	A random forest classifier.
`ensemble.RandomForestRegressor`([…])	A random forest regressor.

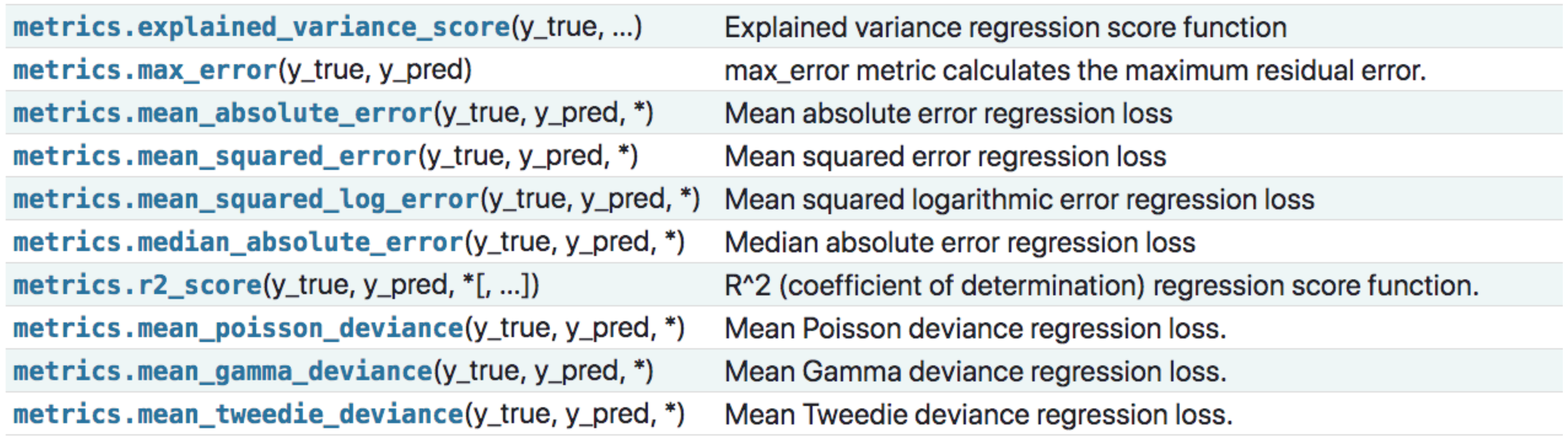
MODULARIZATION

- Evaluation metrics: sklearn.metrics
  - from sklearn.metrics import ****



```
from sklearn.metrics import classification_report, roc_auc_score, precision_score, recall_score, f1_score
```

  - Regression metrics



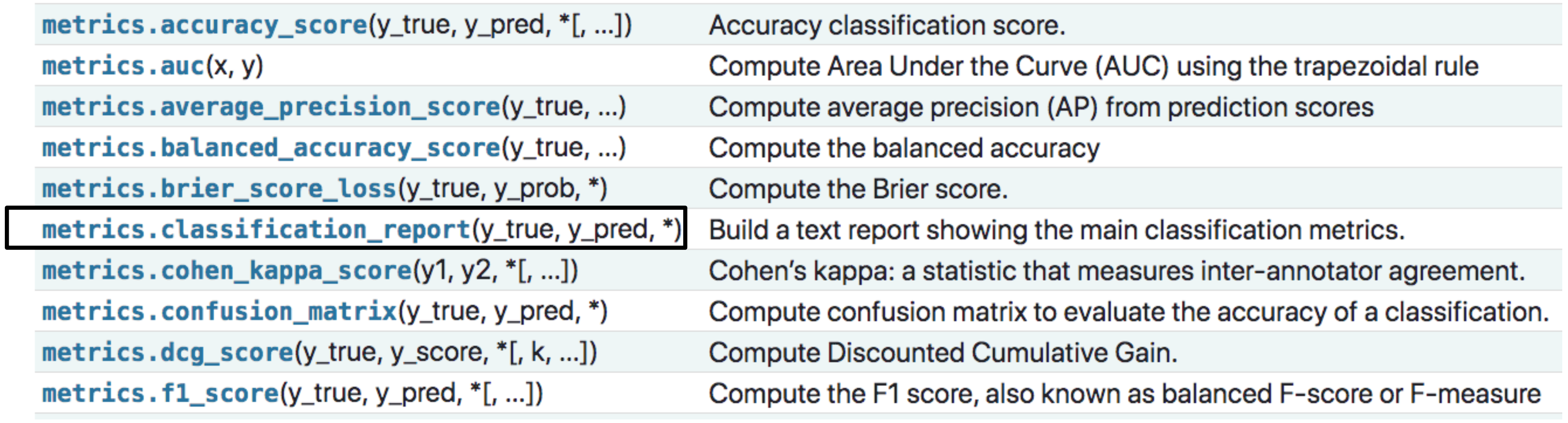
metrics.explained_variance_score(y_true, …)	Explained variance regression score function
metrics.max_error(y_true, y_pred)	max_error metric calculates the maximum residual error.
metrics.mean_absolute_error(y_true, y_pred, *)	Mean absolute error regression loss
metrics.mean_squared_error(y_true, y_pred, *)	Mean squared error regression loss
metrics.mean_squared_log_error(y_true, y_pred, *)	Mean squared logarithmic error regression loss
metrics.median_absolute_error(y_true, y_pred, *)	Median absolute error regression loss
metrics.r2_score(y_true, y_pred, *[, …])	R^2 (coefficient of determination) regression score function.
metrics.mean_poisson_deviance(y_true, y_pred, *)	Mean Poisson deviance regression loss.
metrics.mean_gamma_deviance(y_true, y_pred, *)	Mean Gamma deviance regression loss.
metrics.mean_tweedie_deviance(y_true, y_pred, *)	Mean Tweedie deviance regression loss.

MODULARIZATION

- Evaluation metrics: sklearn.metrics
  - from sklearn.metrics import ****

```
from sklearn.metrics import classification_report, roc_auc_score, precision_score, recall_score, f1_score
```

  - Classification metrics



metrics.accuracy_score(y_true, y_pred, *[, ...])	Accuracy classification score.
metrics.auc(x, y)	Compute Area Under the Curve (AUC) using the trapezoidal rule
metrics.average_precision_score(y_true, ...)	Compute average precision (AP) from prediction scores
metrics.balanced_accuracy_score(y_true, ...)	Compute the balanced accuracy
metrics.brier_score_loss(y_true, y_prob, *)	Compute the Brier score.
metrics.classification_report(y_true, y_pred, *)	Build a text report showing the main classification metrics.
metrics.cohen_kappa_score(y1, y2, *[, ...])	Cohen's kappa: a statistic that measures inter-annotator agreement.
metrics.confusion_matrix(y_true, y_pred, *)	Compute confusion matrix to evaluate the accuracy of a classification.
metrics.dcg_score(y_true, y_score, *[, k, ...])	Compute Discounted Cumulative Gain.
metrics.f1_score(y_true, y_pred, *[, ...])	Compute the F1 score, also known as balanced F-score or F-measure

MODULARIZATION

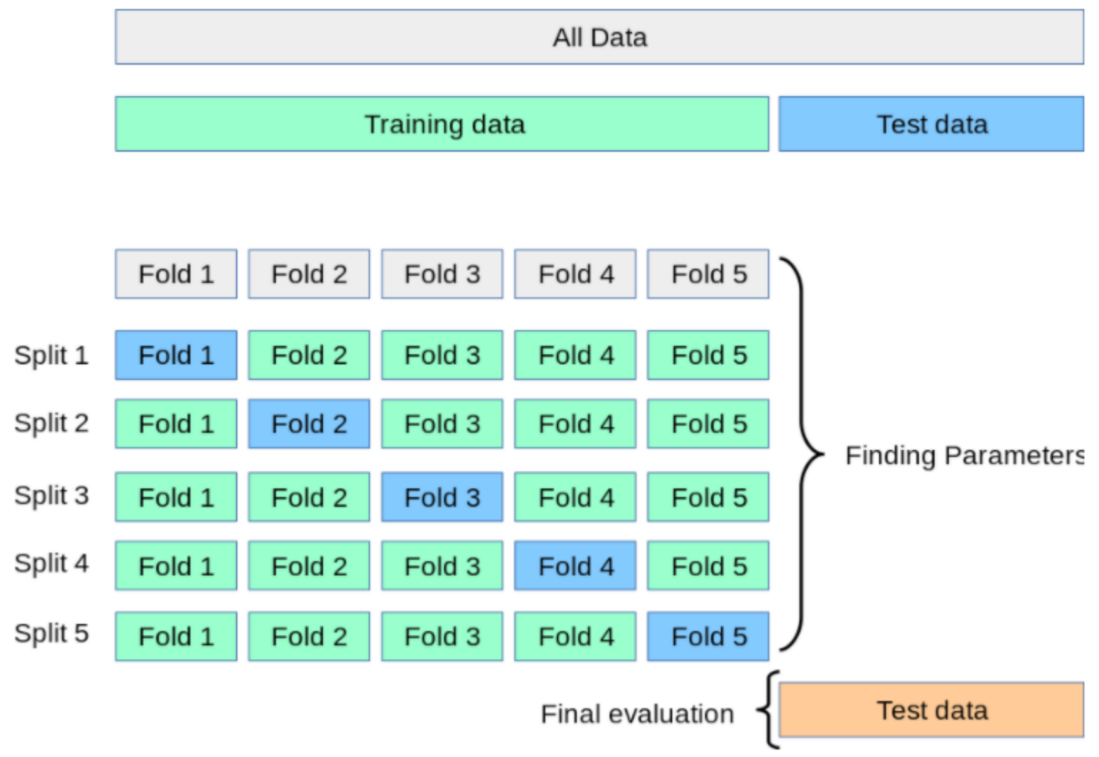
- Other modules (classes)
  - https://scikit-learn.org/stable/modules/classes
  - SVM, MLP etc.

A COMPLETE EXAMPLE

- Predict gender/tissue from gene expression profiles
  - See Jupyter Notebook

CROSS VALIDATION

- Hyper parameter tuning
  - Manually
  - Automatically using scikit-learn
- Splitting function
  - train_test_split()
  - StratifiedKFold()



OTHER EXAMPLES

- Clustering
- PCA
- Lasso
- Other datasets

Thank you!

Q&A