

Learning Scikit-Learn

Qiyang Hu

UCLA Office of Advanced Research Computing

July 15, 2022

Outline

- Learning Scikit-learn



- High-level overview of scikit-learn libraries
 - According to a typical machine learning workflow
 - A lot of colab snippets as examples
 - Titanic challenge in Kaggle (optional)
- Practical usage of scikit-learn
 - Deep learning and scikit-learn
 - Scikit-learn extension libraries

Today

- High-performance machine learning using scikit-learn

- Overview of performance issues in machine learning
- Making the computation faster
- Processing large dataset

What can/can't be expected in the series?

 CAN	 CAN'T
<ul style="list-style-type: none">• Review on Machine learning workflows	<ul style="list-style-type: none">• Introduction to various Machine Learning models
<ul style="list-style-type: none">• A <i>BIG</i> picture on scikit-learn's features, functions & components	<ul style="list-style-type: none">• Discussions on the details of specific scikit-learn function interfaces
<ul style="list-style-type: none">• Providing handy examples as demos (mainly for studying <i>after</i> the class)	<ul style="list-style-type: none">• Line-by-line explanation on every demo code
<ul style="list-style-type: none">• High-level introduction on high performance machine learning	<ul style="list-style-type: none">• Lectures on detailed mechanism and implementations of HPML.

Knowing Scikit-Learn in minutes

- A Python machine learning framework
 - Library built on numpy, scipy, matplotlib
 - Started in 2007, publicly released in 2010
 - Is currently maintained by volunteers
- Installation/Loading
 - `conda install -c intel scikit-learn`
 - On H2: `module load anaconda3`
`conda activate sklearn`
 - Using Google Colab
- Designed for easy-to-use productions
 - Simplicity
 - Qualitative code
 - Performance
 - Elegant APIs
 - Excellent docs: <https://scikit-learn.org>

```
import sklearn
```

```
# 2 samples, 3 features  
X = [[ 1,  2,  3],  
      [11, 12, 13]]
```

Data

```
# classes of each sample  
y = [0, 1]
```

Modeling

```
from sklearn.ensemble import RandomForestClassifier
```

```
clf = RandomForestClassifier(random_state=0)
```

```
clf.fit(X, y)
```

```
# predict classes of the training data  
clf.predict(X)
```

```
# predict classes of new data  
clf.predict([[4, 5, 6], [14, 15, 16]])
```

Predicting

A surrealist illustration depicting a man and a woman standing on a small, floating island of green grass. The island is supported by a dense layer of soldiers in khaki uniforms, who are holding up the grass with their arms. The soldiers are submerged in a dark red sea. In the background, a lighthouse sits on a small island, and a beach with a house is visible under a blue sky. The overall scene suggests that the couple's peaceful life is sustained by the labor and sacrifice of many others.

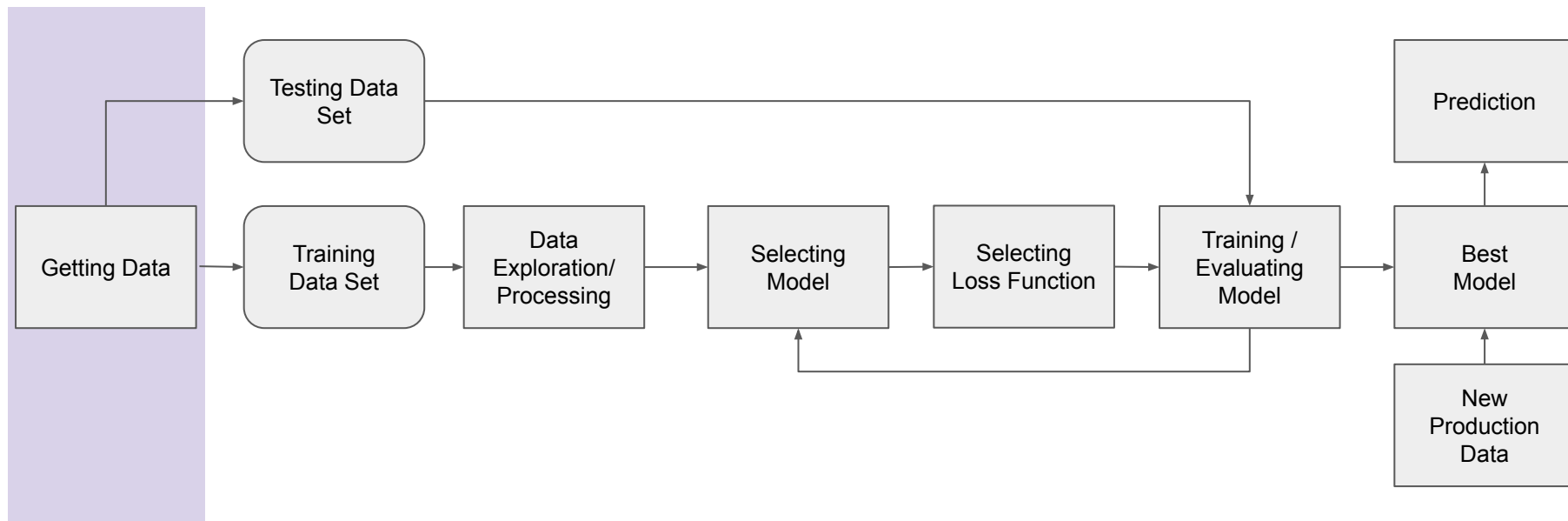
Me using
scikit-learn

All the mathematics,
research and coding skills

Outline

- Learning Scikit-learn
 - High-level overview of scikit-learn libraries
 - According to a typical machine learning workflow
 - A lot of colab snippets as examples
 - Titanic challenge in Kaggle (optional)
 - Practical usage of scikit-learn
 - Deep learning and scikit-learn
 - Scikit-learn extension libraries
- High-performance machine learning using scikit-learn
 - Overview of performance issues in machine learning
 - Making the computation faster
 - Processing large dataset

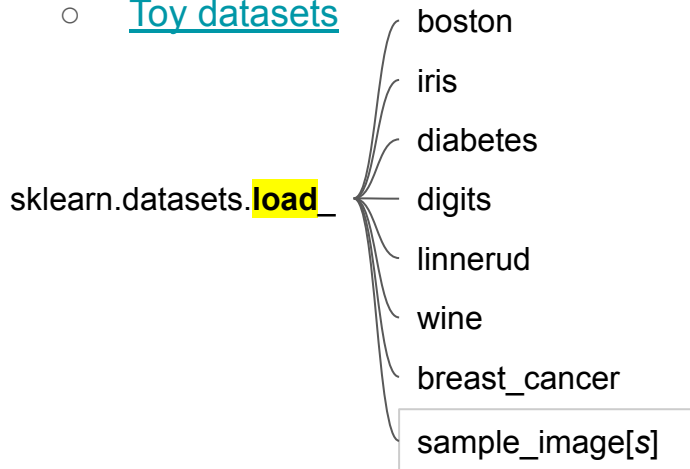
Simplified workflow for a machine learning project



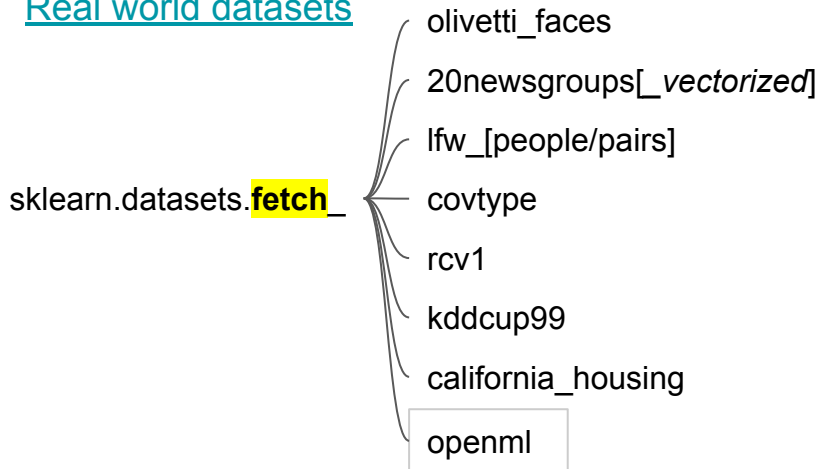
Data input and data loader

- Data format can be input directly as:
 - Dense data: `numpy.ndarray`
 - Sparse data: `scipy.sparse.matrix`
- Data can be loaded from standard datasets:

- Toy datasets



- Real world datasets



Loaders and
Fetchers
returns a
dictionary-like
bunch object

```
>>> b = Bunch(a=1, b=2)
>>> b['b']
2
>>> b.b
2
>>> b.a = 3
>>> b['a']
3
>>> b.c = 6
>>> b['c']
6
```


Data Generator

sklearn.datasets.**make_**

blob

classification

gaussian_quantiles

hastie_10_2

circles

moons

multilabel_classification

biclusters

checkerboard

regression

friedman[1/2/3]

sparse_uncorrelated

s_curve

swiss_roll

low_rank_matrix

sparse_coded_signal

spd_matrix

sparse_spd_matrix

```
(n_samples=100, n_features=2, *,  
centers=None, cluster_std=1.0,  
center_box=(- 10.0, 10.0,  
shuffle=True, random_state=None,  
return_centers=False)
```

For classification and clustering

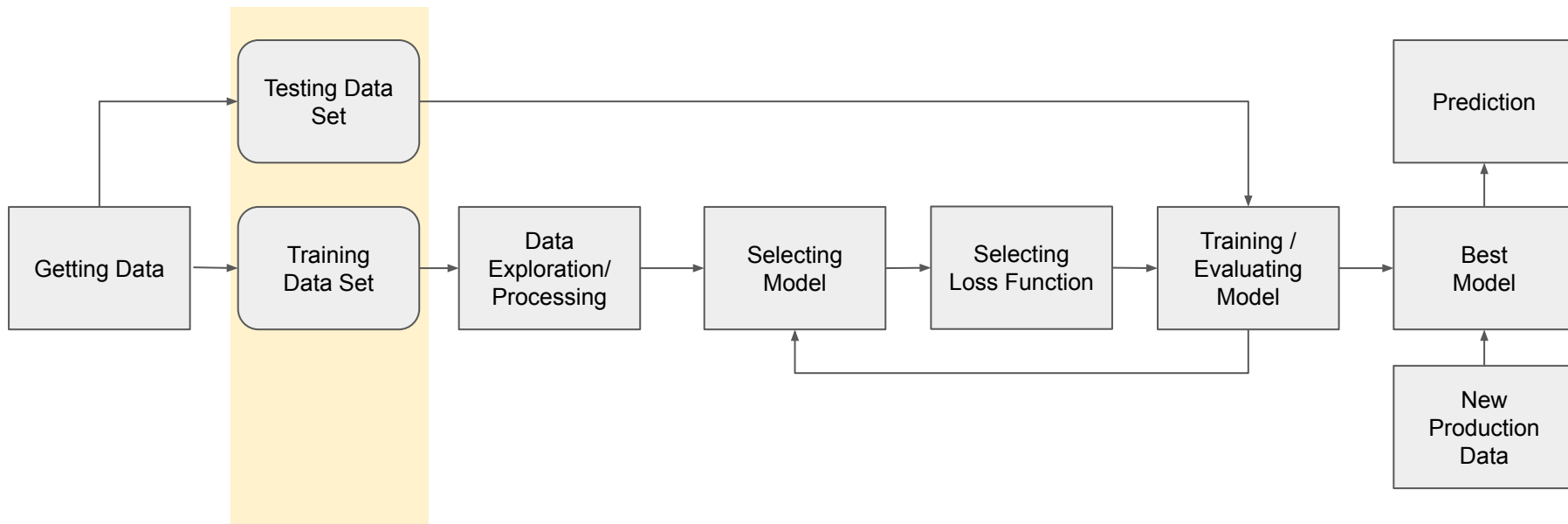
For regression

For manifold learning

For decomposition

bit.ly/1sk1_01

Workflow for a machine learning project



Split training and testing dataset

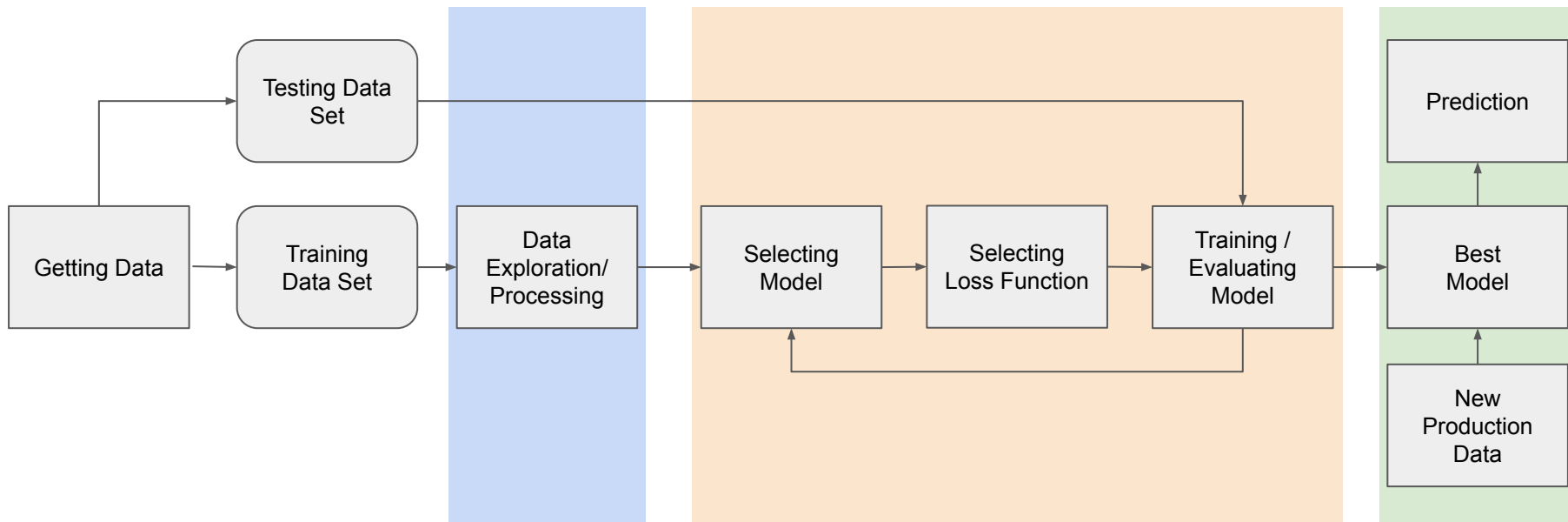
- Essential for an unbiased evaluation of prediction performance
 - Process related with model evaluation and selection
 - Internally, scikit-learn uses cross-validation iterators to split
- Multiple splitting methods
 - Stratified splitting
 - Group splitting
 - Time series splitting
 - Predefined splitting
- Sklearn's `train_test_split`
 - A wrapper around `ShuffleSplit`
 - Only allows for stratified splitting
 - As a base for the default cross-validations

```
>>> import numpy as np
>>> from sklearn.model_selection import train_test_split
>>> from sklearn import datasets
>>> from sklearn import svm

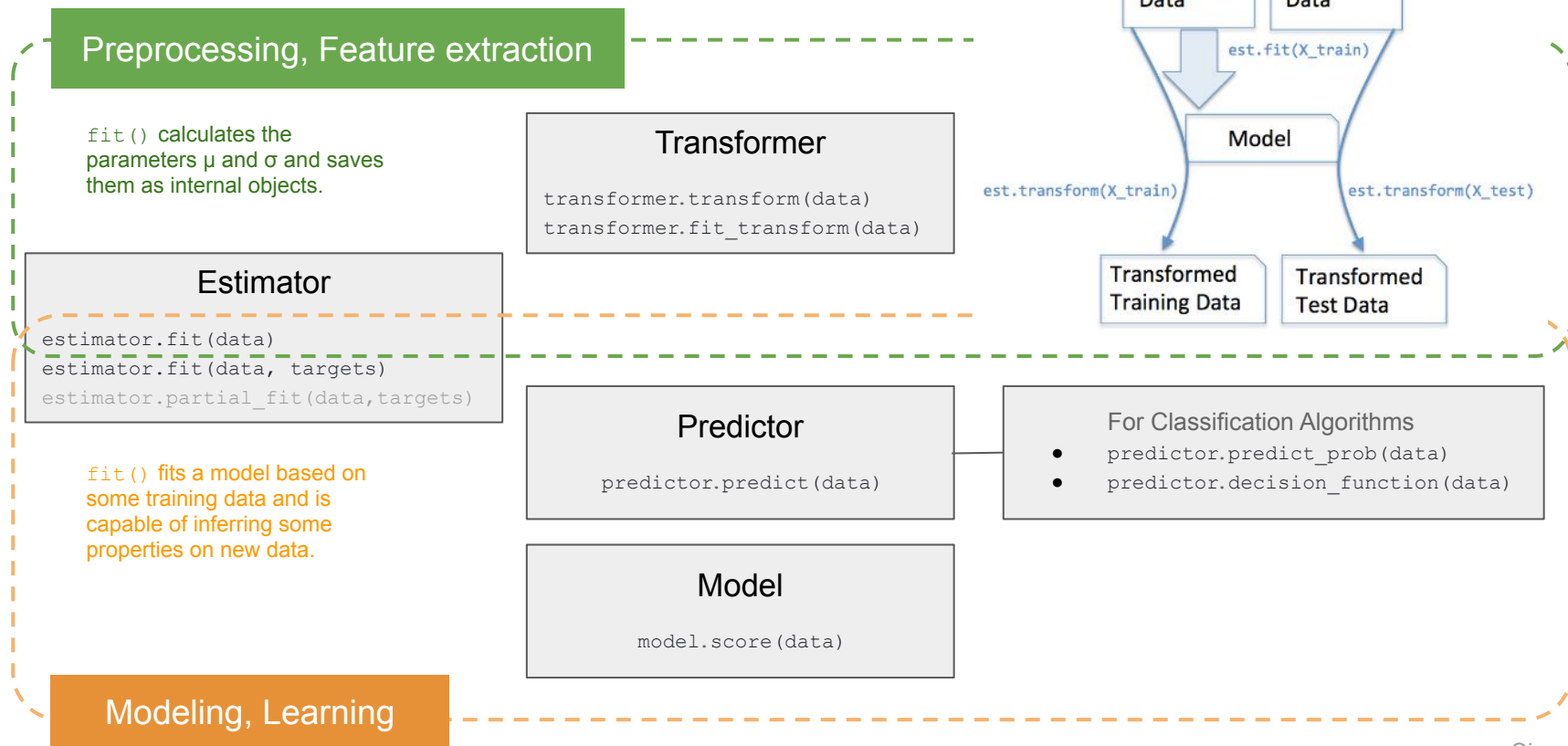
>>> X, y = datasets.load_iris(return_X_y=True)
>>> X.shape, y.shape
((150, 4), (150,))
>>> X_train, X_test, y_train, y_test = train_test_split(
...     X, y, test_size=0.4, random_state=0)

>>> X_train.shape, y_train.shape
((90, 4), (90,))
>>> X_test.shape, y_test.shape
((60, 4), (60,))
```

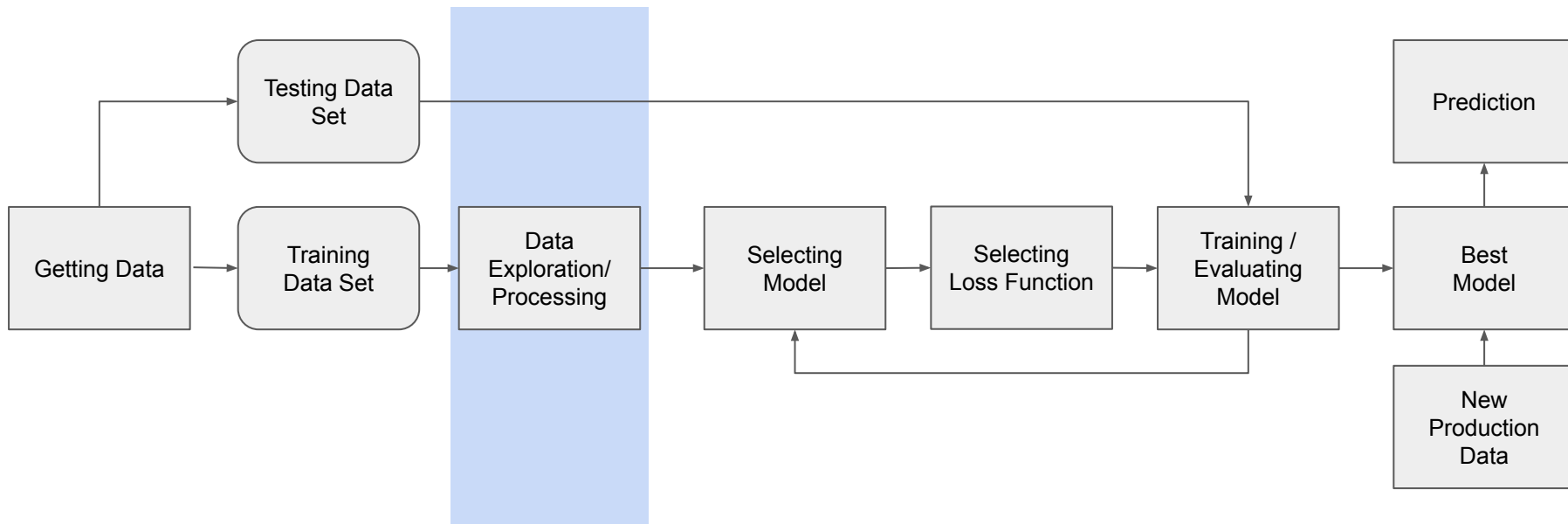
Workflow for a machine learning project



Core objects (estimators)



Workflow for a machine learning project



Preprocessing:

```
from sklearn.preprocessing import  
StandardScaler  
sc = StandardScaler()  
sc.fit_transform(X_train)  
sc.transform(X_test)
```

sklearn.preprocessing.

[StandardScaler](#) / RobustScaler
MinMaxScaler / MaxAbsScaler
KernelCenterer
QuantileTransformer
PowerTransformer
normalize
Normalizer
OrdinalEncoder/LabelEncoder
OneHotEncoder
KBinsDiscretizer
Binarizer
FunctionTransformer
PolynomialFeatures

Standardization, or mean removal
and variance scaling

Non-linear transformation

Normalization

Encoding categorical features

Discretization

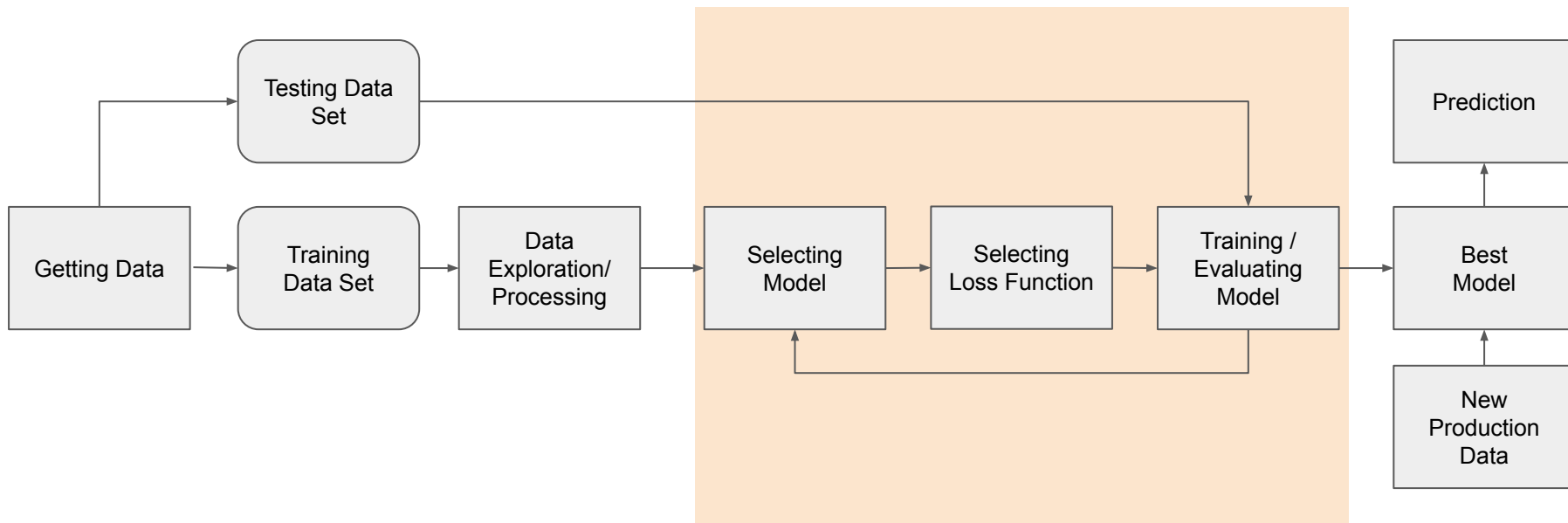
Custom transformers

sklearn.imput.

SimpleImputer
IterativeImputer
KNNImputer
MissingIndicator

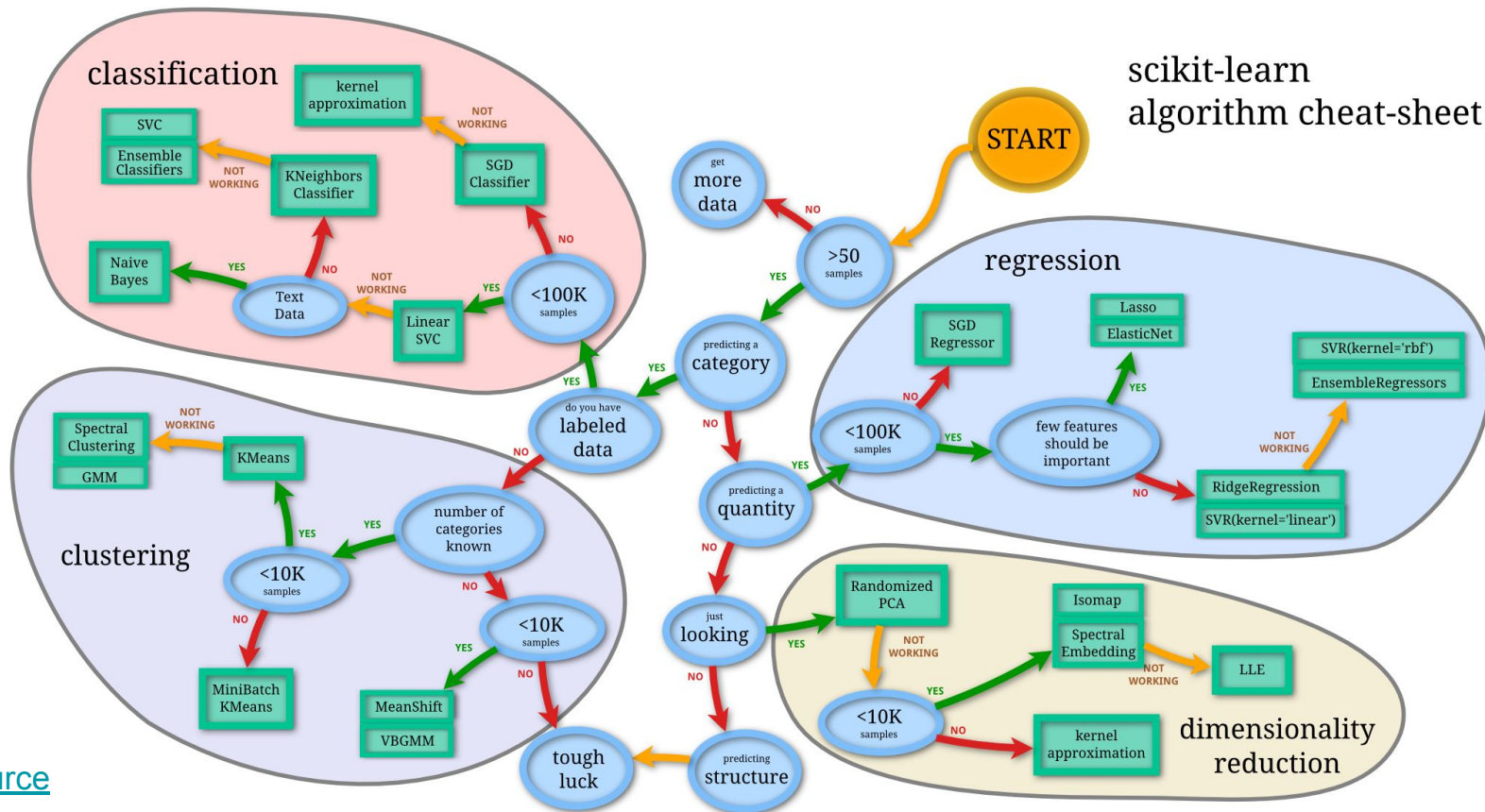
Imputation of missing values

Workflow for a machine learning project



Choosing the right estimator (algorithm)

scikit-learn
algorithm cheat-sheet



[Source](#)

Pseudo-code template for modeling and learning

```
from sklearn. {  
    linear_model  
    svm  
    tree  
    naive_bayes  
    multioutput  
    ensemble  
    cluster  
    decomposition  
    ...  
}
```

```
model = SpecModel( hyperparameter )
```

```
model.fit( X, y )
```

```
y_pred = model.predict( X_new )
```

```
s = model.score( X_new )
```

import

SpecModel

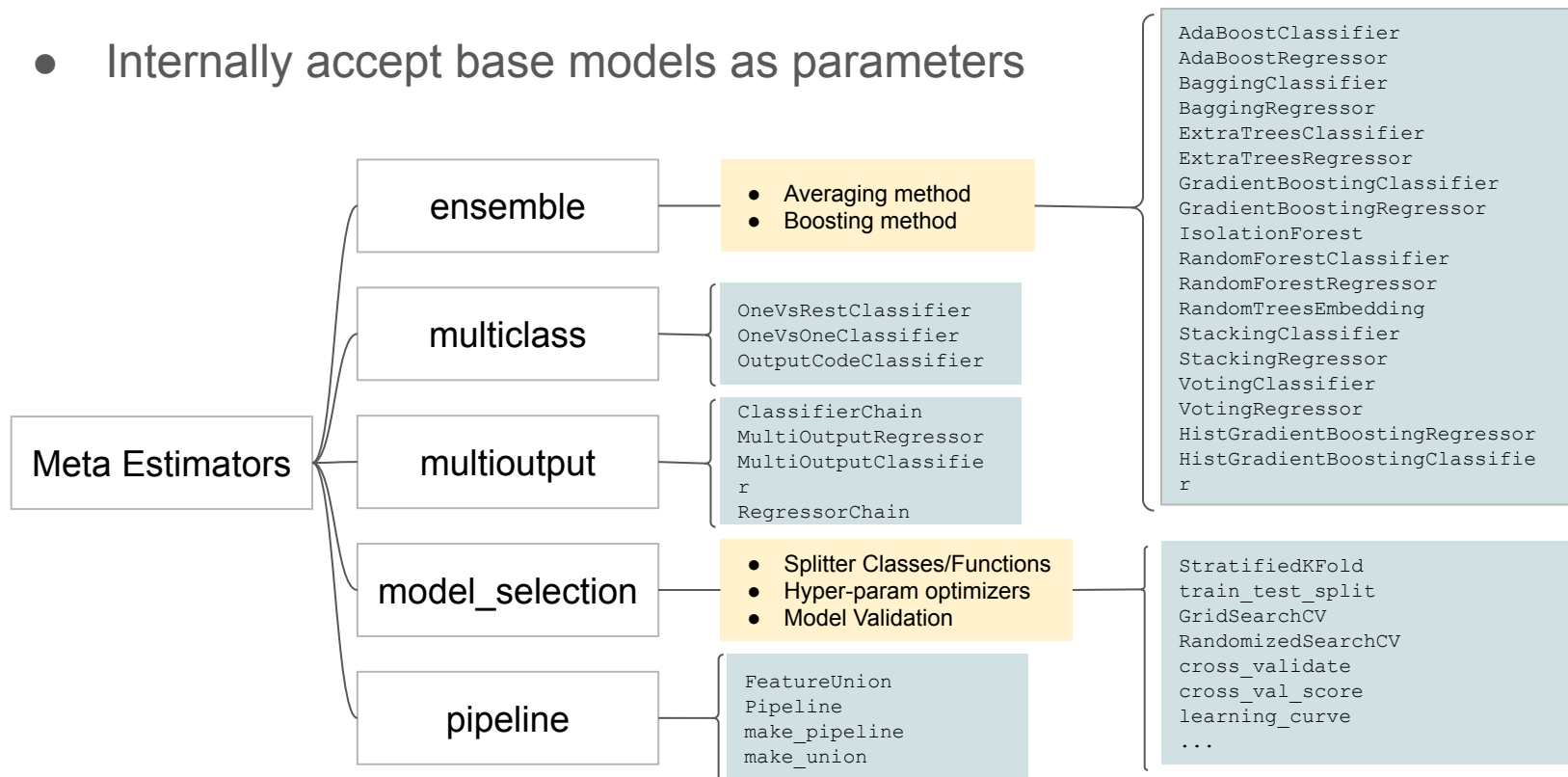
```
penalty='l2', tol=0.0001, C=0.1,  
fit_intercept=True,  
solver='liblinear', max_iter=100,  
multi_class='ovr', n_jobs=1, ...
```

LogisticRegression

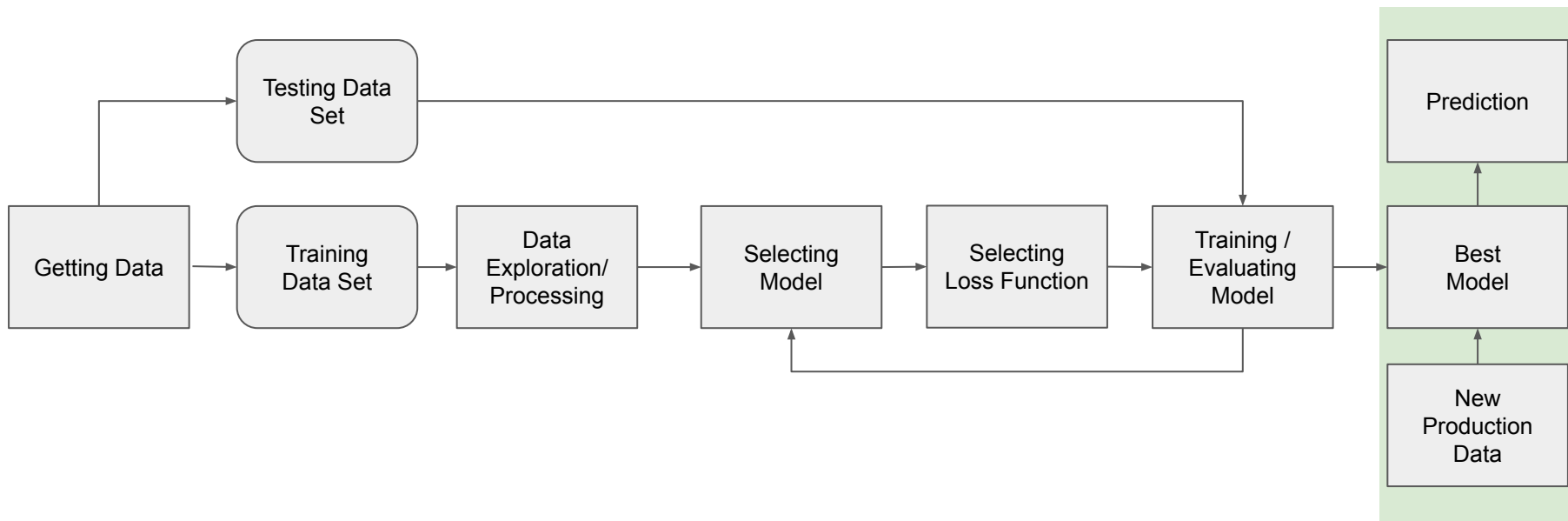
```
LogisticRegressionCV  
PassiveAggressiveClassifier  
Perceptron  
RidgeClassifier  
RidgeClassifierCV  
SGDClassifier  
LinearRegression  
Ridge  
RidgeCV  
SGDRegressor  
ElasticNet  
ElasticNetCV  
Lars  
LarsCV  
Lasso  
LassoCV  
LassoLars  
LassoLarsCV  
LassoLarsIC  
OrthogonalMatchingPursuit  
OrthogonalMatchingPursuitCV  
ARDRegression  
BayesianRidge  
PoissonRegressor  
GammaRegressor  
HuberRegressor  
RANSACRegressor  
...
```

Meta-estimator: as an assembly of base estimators

- Internally accept base models as parameters



Workflow for a machine learning project



Model persistence (saving/restoring a trained model)

- Python's built-in serialization:
 - Using `pickle` or `joblib`: dump and load
 - Custom transformers in Pipeline cannot be serialized by pickle or joblib
 - Consider using Neuralxle's module to [save custom pipeline](#) in step wise
 - Pickled model better to be deployed using containers to avoid portability issues
- Other exporting formats
 - Open Neural Network Exchange (ONNX)
 - [sklearn-onnx](#)
 - Predictive Model Markup Language (PMML)
 - [sklearn2pmml](#)

Outline

- Learning Scikit-learn
 - High-level overview of scikit-learn libraries
 - According to a typical machine learning workflow
 - A lot of colab snippets as examples
 - Titanic challenge in Kaggle (optional)
 - Practical usage of scikit-learn
 - Deep learning and scikit-learn
 - Scikit-learn extension libraries
- High-performance machine learning using scikit-learn
 - Overview of performance issues in machine learning
 - Making the computation faster
 - Processing large dataset

Titanic Kaggle Challenge

To predict the survival or the death of a given passenger in Titanic

- Titanic Facts:
 - Survivors
 - 492 passengers
 - 214 crews
 - Victims
 - 832 passengers
 - 685 crews
 - Death causes: drowning, hypothermia, injury, suicide, ...
 - [List of deaths](#)



bit.ly/1sk1_02

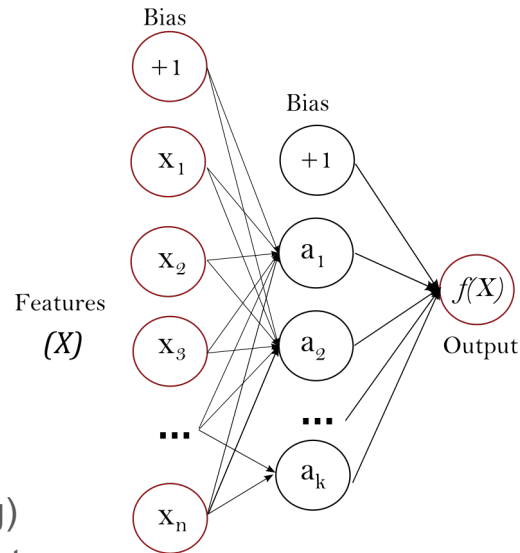
Outline

- Learning Scikit-learn
 - High-level overview of scikit-learn libraries
 - According to a typical machine learning workflow
 - A lot of colab snippets as examples
 - Titanic challenge in Kaggle (optional)
 - Practical usage of scikit-learn
 - Deep learning and scikit-learn
 - Scikit-learn extension libraries
- High-performance machine learning using scikit-learn
 - Overview of performance issues in machine learning
 - Making the computation faster
 - Processing large dataset

Deep Learning and Scikit-Learn

- Neural networks in scikit-learn

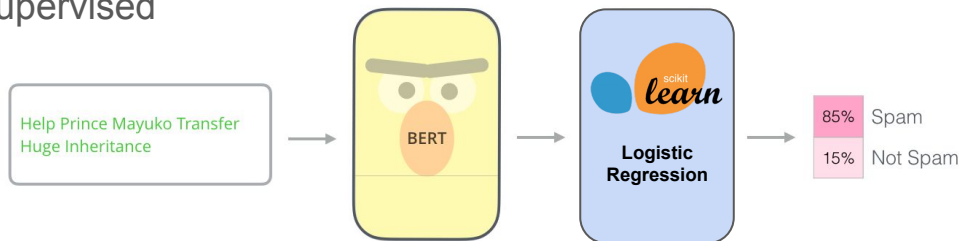
- Multi-layer Perceptron: MLPRegressor, MLPClassifier
 - `from sklearn.neural_network import MLPClassifier`
 - `MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5, 2))`
- Not versatile as Tensorflow and Pytorch, **but**
 - Much simpler and straightforward (esp. for early-stopping)
 - Directly support sparse data as input, saving memory a lot
 - Natively support partial-fit (will discuss in next talk)!



bit.ly/3PuFTHw

- Works together with modern exotic deep learning models

- Large NLP models: pre-trained, semi-supervised
- Scikit-Learn for supervised fine-tuning
 - As a top classifier layer
 - Defining the workflow



Scikit-learn extension libraries

- Libraries adopting scikit-learn functionalities
 - Data formats: [sklearn_pandas](#), [sklear_xarray](#), ...
 - Auto-ML: [auto-sklearn](#), [Featuretools](#), [Neuraxle](#), ...
 - Model visualization: [dtreeviz](#), [eli5](#), ...
 - Model selection: [scikit-optimize](#), [sklearn-deap](#), ...
 - Model export: [onnxmltools](#), [sklearn2pmmml](#), ...
 - Parallelization: [sk-dist](#)
 - Plotting: [scikit-plot](#)
- Libraries compatible with scikit-learn interfaces
 - Time-series models: [tslearn](#), [sktime](#), [seglearn](#), ...
 - Deep learning: [keras](#), [skorch](#), ...
 - Other regression/classification: [xgboost](#), [ML_Ensemble](#), [gplearn](#), ...
 - Decomposition and clustering: [lda](#), [hdbscan](#), ...

SciKits means a huge family of SciPy libraries

- scikit-learn
- scikit-opt
- scikit-image
- scikit-sparse
- scikit-statsmodels
- ...

Outline

- Learning Scikit-learn
 - High-level overview of scikit-learn libraries
 - According to a typical machine learning workflow
 - A lot of colab snippets as examples
 - Titanic challenge in Kaggle (optional)
 - Practical usage of scikit-learn
 - Deep learning and scikit-learn
 - Scikit-learn extension libraries
- High-performance machine learning using scikit-learn
 - Overview of performance issues in machine learning
 - Making the computation faster
 - Processing large dataset

**See you
Next Friday!**

bit.ly/1sk1_survey1