Learning Scikit-Learn

Qiyang Hu

UCLA IDRE/OARC Workshop

February 25th, 2021

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

Data

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

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

Modeling

```
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(random_state=0)

clf.fit(X, y)
```

Predicting

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

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

Outline

- High-level overview of scikit-learn libraries
  - According to a typical machine learning workflow
  - A lot of colab snippets as examples
- Demo for working on a fun machine learning project
  - Titanic challenge in Kaggle
- Discussion on advanced topics
  - Boosting scikit-learn performance
  - Scikit-learn extension libraries
  - High-performance machine learning

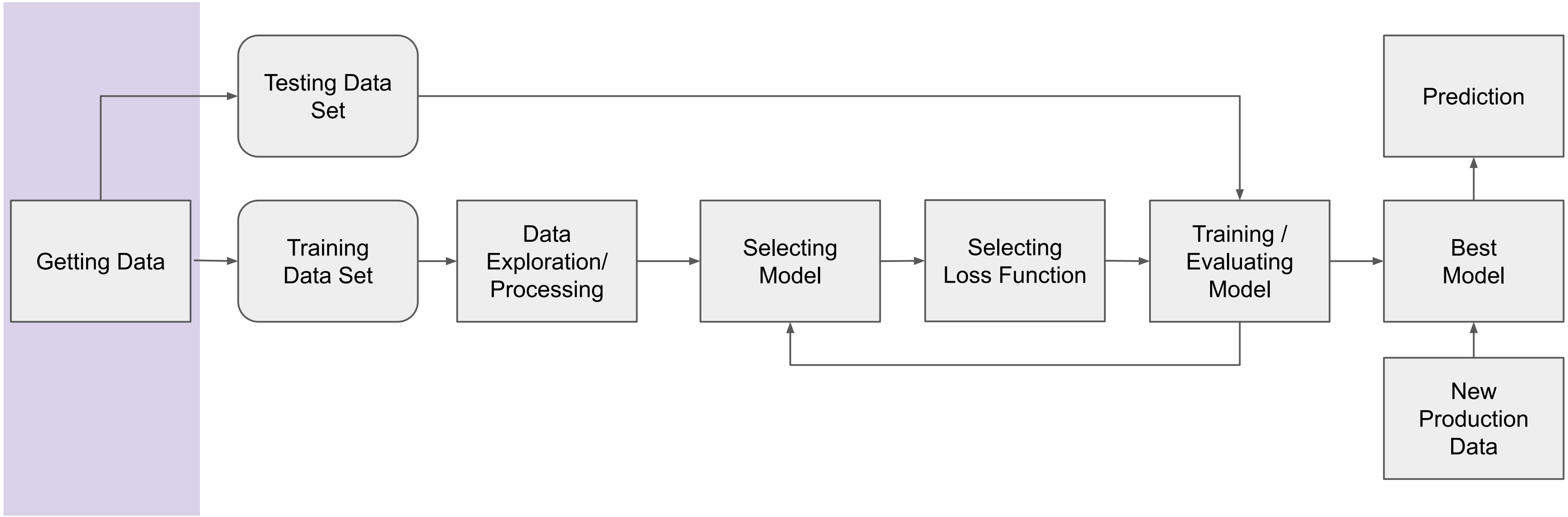
What can/can’t be expected in this class?

✓ CAN	✗ CAN’T
• Review on Machine learning workflows	• Introduction to various Machine Learning models
• A ***<u>BIG</u>*** picture on scikit-learn’s features, functions & components	• Discussions on the details of specific scikit-learn function interfaces
• Providing handy examples as demos (probably for studying *after* the class)	• Line-by-line explanation on every demo code
• High-level introduction on some advanced topics	• A lecture on high-performance machine learning

Outline

- High-level overview of scikit-learn libraries
  - According to a typical machine learning workflow
  - A lot of colab snippets as examples
- Demo for working on a fun machine learning project
  - Titanic challenge in Kaggle
- Discussion on advanced topics
  - Boosting scikit-learn performance
  - Scikit-learn extension libraries
  - High-performance machine learning

Workflow for a machine learning project



Data input and 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

    sklearn.datasets.**load**_
    - boston
    - iris
    - diabetes
    - digits
    - linnerud
    - wine
    - breast_cancer
    - sample_image[*s*]
  - Real world datasets

    sklearn.datasets.**fetch**_
    - olivetti_faces
    - 20newsgroups[_*vectorized*]
    - lfw_[people/pairs]
    - covtype
    - rcv1
    - kddcup99
    - california_housing
    - openml

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

```
(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

- regression
- friedman[*1/2/3*]
- sparse_uncorrelated

For regression

- s_curve
- swiss_roll

For manifold learning

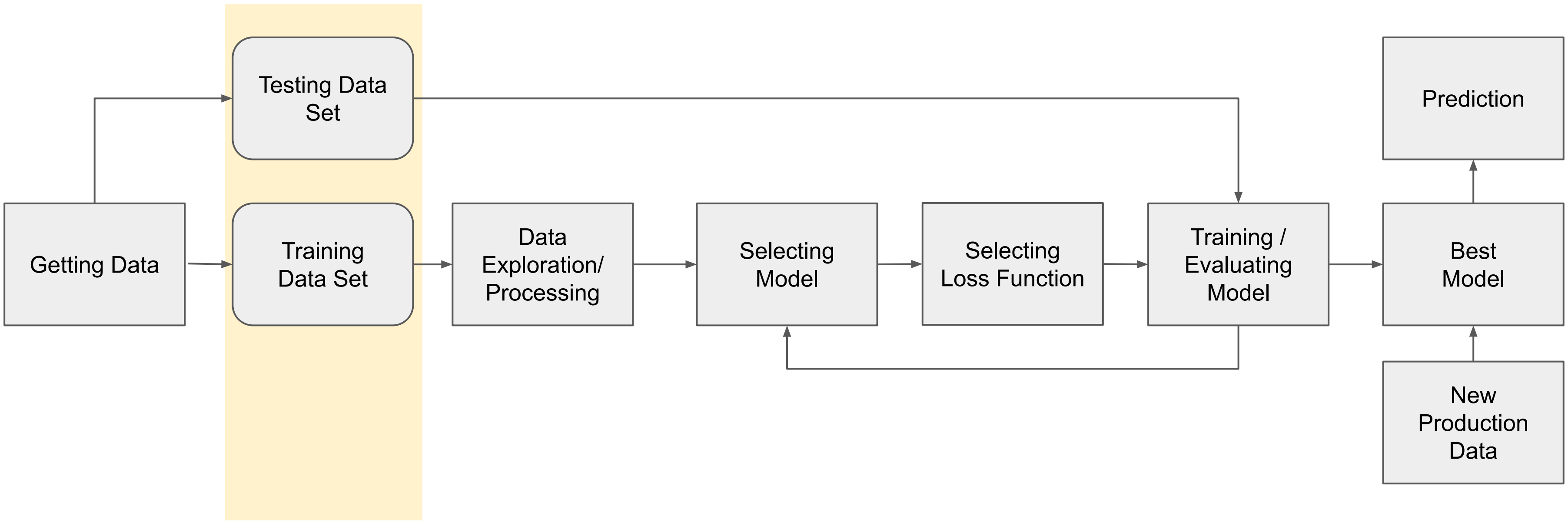
- low_rank_matrix
- sparse_coded_signal
- spd_matrix
- sparse_spd_matrix

For decomposition

Qiyang Hu

bit.ly/lskl_01

Workflow for a machine learning project

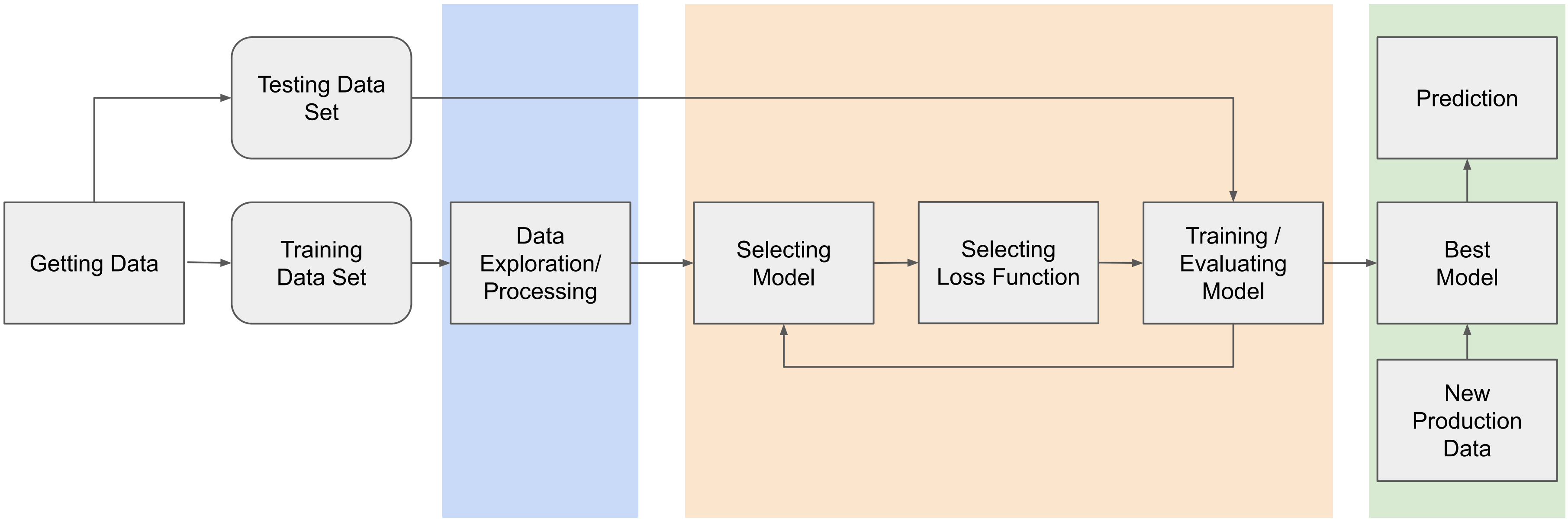


Split training and testing dataset

- Essential for an unbiased evaluation of prediction performance
  - Process related with model evaluation and selection
- Sklearn has helper function `train_test_split` to randomly split
  - A wrapper around ShuffleSplit
  - Only allows for stratified splitting
  - Cannot account for groups
  - 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



Qiyang Hu

Core objects (estimators)

Preprocessing, Feature extraction

`fit()` calculates the parameters μ and σ and saves them as internal objects.

Transformer

```
transformer.transform(data)
transformer.fit_transform(data)
```

`transform(X_test)` using the calculated params apply the transformation to a particular dataset.

`fit_transform(X_train)` joins the fit() and transform() method for transformation of dataset.

Estimator

```
estimator.fit(data)
estimator.fit(data, targets)
```

`fit()` fits a model based on some training data and is capable of inferring some properties on new data.

Predictor

```
predictor.predict(data)
```

For Classification Algorithms

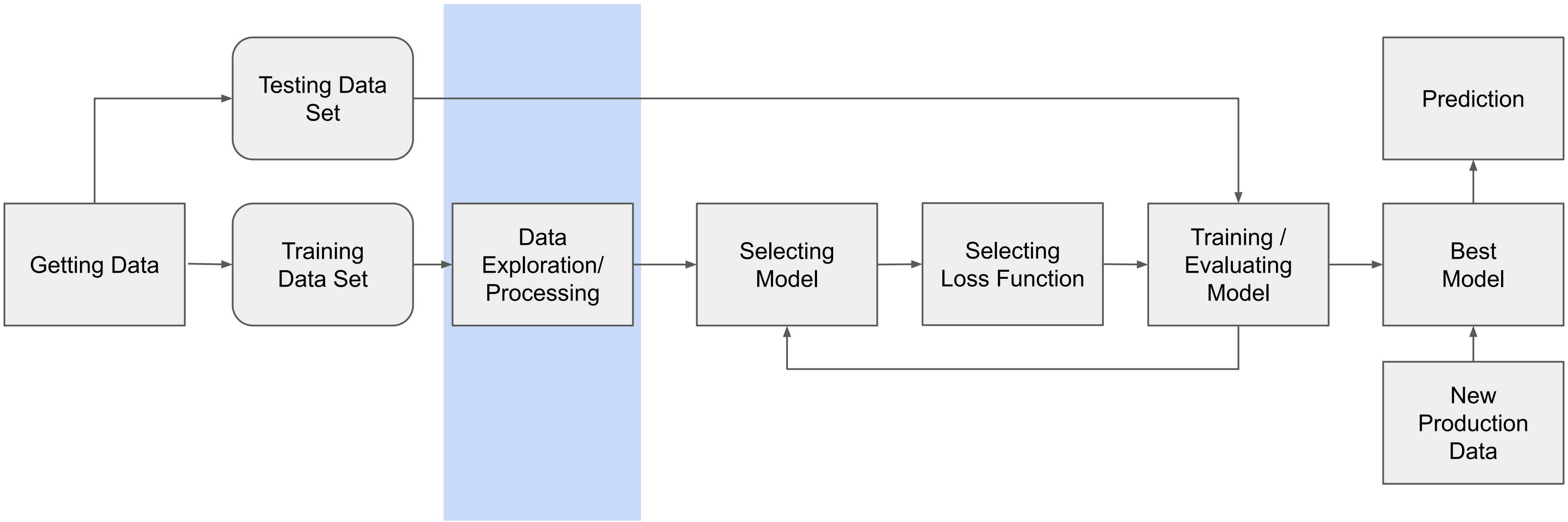
- `predictor.predict_prob(data)`
- `predictor.decision_function(data)`

Model

```
model.score(data)
```

Modeling, Learning

Workflow for a machine learning project



Preprocessing:

sklearn.**preprocessing**.

- StandardScaler / RobustScaler
- MinMaxScaler / MaxAbsScaler
- KernelCenterer

Standardization, or mean removal and variance scaling

- QuantileTransformer
- PowerTransformer

Non-linear transformation

- normalize
- Normalizer

Normalization

- OrdinalEncoder/LabelEncoder
- OneHotEncoder

Encoding categorical features

- KBinsDiscretizer
- Binarizer

Discretization

- FunctionTransformer
- PolynomialFeatures

Custom transformers

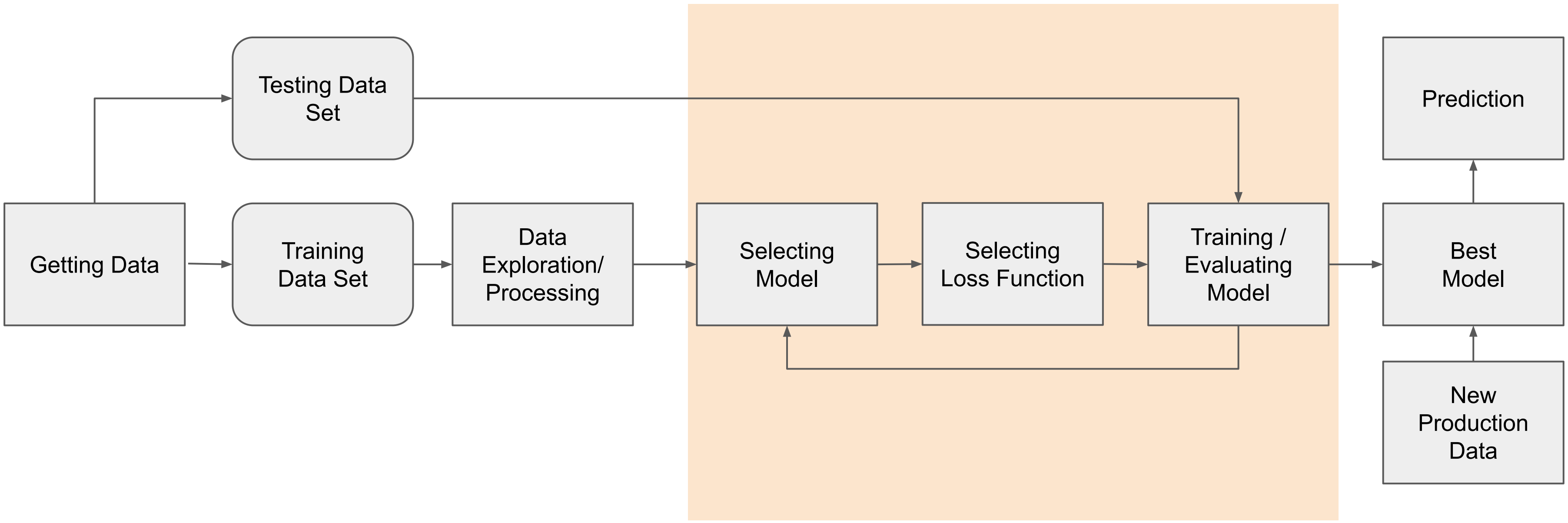
```
from sklearn.preprocessing import
StandardScaler
sc = StandardScaler()
sc.fit_tranform(X_train)
sc.transform(X_test)
```

sklearn.**imput**.

- SimpleImputer
- IterativeImputer
- KNNImputer
- MissingIndicator

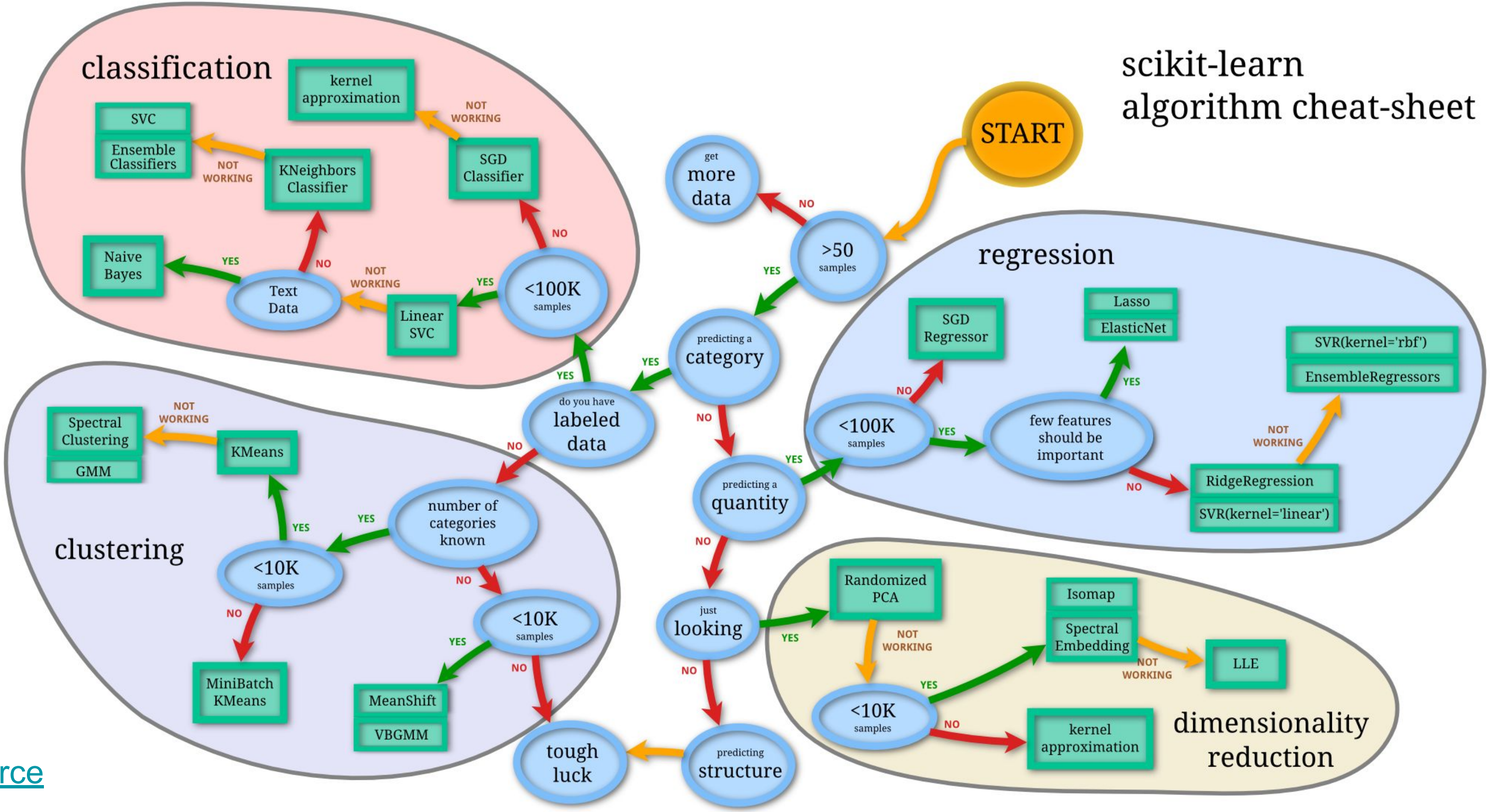
Imputation of missing values

Workflow for a machine learning project



Qiyang Hu

Choosing the right estimator (algorithm)



[Source](#)

Pseudo-code template for modeling and learning

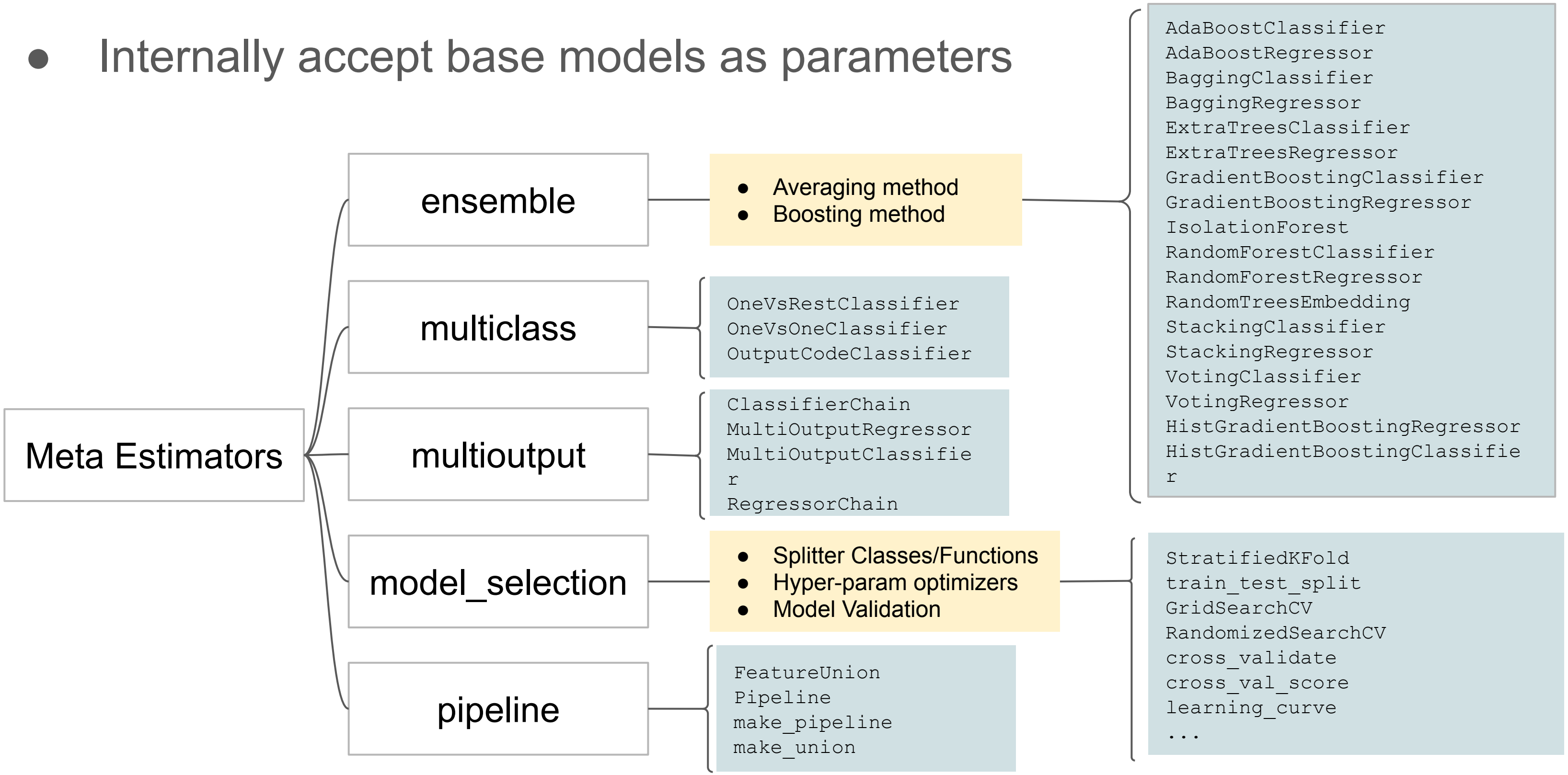
```
                 { linear_model     }
                 { svm              }
                 { tree             }
from sklearn.    { naive_bayes      }  import SpecModel
                 { multioutput      }
                 { ensemble         }
                 { cluster          }
            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

model = SpecModel( hyperparameter )

model.fit( X, y )

y_pred = model.predict( X_new )

s = model.score( X_new )
```

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

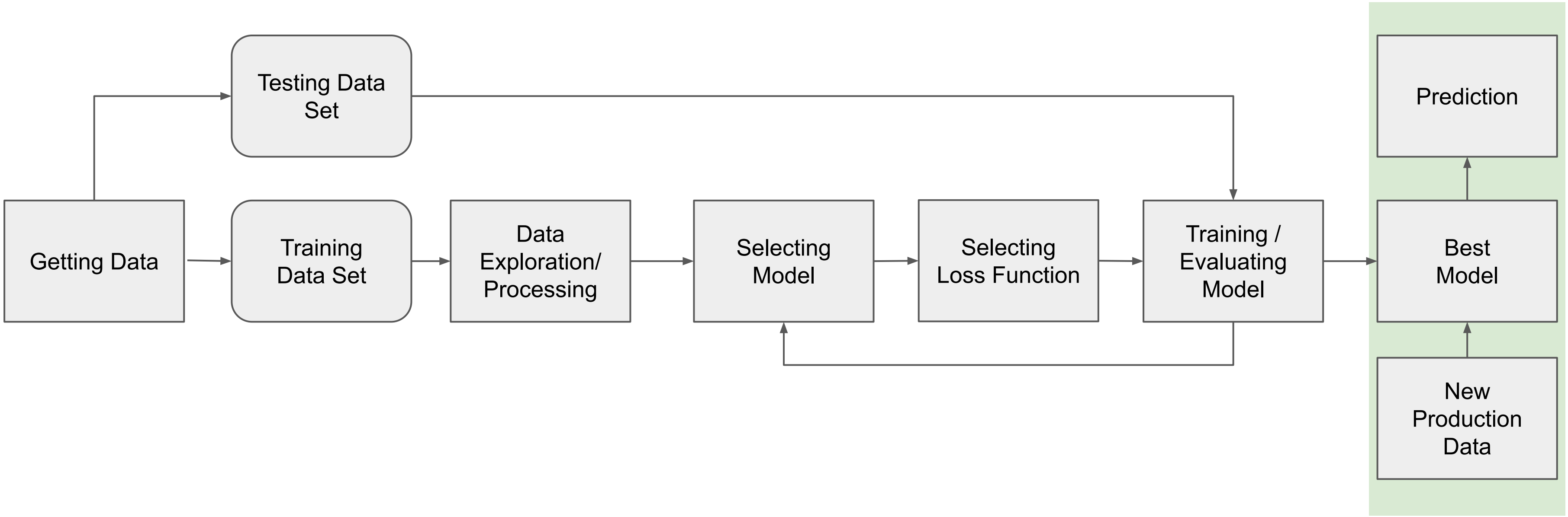
```
LogisticRegression
LogisticRegressionCV
PassiveAggressiveClassifier
Perceptron
RidgeClassifier
RidgeClassifierCV
SGDClassifier
LinearRegression
Ridge
RidgeCV
SGDRegressor
ElasticNet
ElasticNetCV
Lars
LarsCV
Lasso
LassoCV
LassoLars
LassoLarsCV
LassoLarsIC
OrthogonalMatchingPursuit
OrthogonalMatchingPursuitCV
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



Qiyang Hu

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

- High-level overview of scikit-learn libraries
  - According to a typical machine learning workflow
  - A lot of colab snippets as examples
- Demo for working on a fun machine learning project
  - Titanic challenge in Kaggle
- Discussion on advanced topics
  - Boosting scikit-learn performance
  - Scikit-learn extension libraries
  - High-performance machine learning

Titanic Kaggle Challenge

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

- Titanic Facts:
  - Survivors
    - 492 passagers
    - 214 crews
  - Victims
    - 832 passagers
    - 685 crews
    - Death causes: drowning, hypothermia, injury, suicide, …
    - [List of deaths]

bit.ly/lskl_02

Outline

- High-level overview of scikit-learn libraries
  - According to a typical machine learning workflow
  - A lot of colab snippets as examples
- Demo for working on a fun machine learning project
  - Titanic challenge in Kaggle
- Discussion on advanced topics
  - Boosting scikit-learn performance
  - Scikit-learn extension libraries
  - High-performance machine learning

Basic performance tips for scikit-learn projects

- Profiling your Python code
  - Using line_profiler, memory_profiler
- Using Numpy and Scipy as much as possible to replace nested for loops
  - [Performance tips for numpy and scipy]
- Writing cython wrappers for well-maintained C/C++ algorithm implementation
  - example: liblinear, libsvm used in Logistic Regression and SVM models.
- Multiple-core parallelism:
  - Specifying `n_jobs` for model training, evaluation, hyperparameter tuning
  - Using OpenMP functions through Cython imported by `cimport openmp`
  - Using `joblib.Parallel`
    - joblib is a set of tools to provide lightweight pipelining in Python.
    - Already used in some sklearn classes (ElasticNet, SGDClassifier, …)
    - Various parallel backends: `locky`, `multiprocessing`, `threading`, `dask`, `ray`, ...

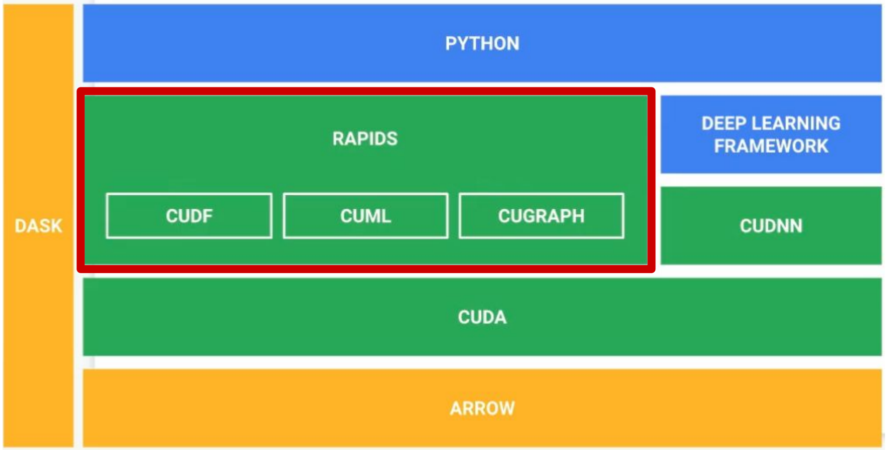
Scikit-learn extension libraries

- 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, sklearn2pmml, …
  - 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, ...

Libraries *nothing* related with scikit-learn
• scikit-opt

High-performance machine learning

- Scikit-learn alone doesn't have support to GPU or TPU.
- Option 1: RAPIDS software libraries from nvidia
  - Exactly same APIs
  - Kernels rewritten by CUDA
    - ➢ *Numpy -> CuPy*
    - ➢ pandas -> cuDF
    - ➢ scikit-learn -> cuML
    - ➢ networkx -> cuGraph
  - Dask + RAPIDS + blazingSQL, RAPIDS + Spark, xgboost, ...



- Option 2: JAX
  - JAX: compile NumPy using XLA, on GPUs and TPUs for high-performance machine learning.
  - `sklearn-jax-kernel` is in an early stage.

High-performance machine learning (cont'd)

- Option 3: using [daal4py]
  - Python APIs to Intel's OneDAL
  - Provide alternative estimators
  - Acceleration without code change:
    - **Intel** CPU optimization patching

```
from daal4py.sklearn import patch_sklearn
patch_sklearn()
```

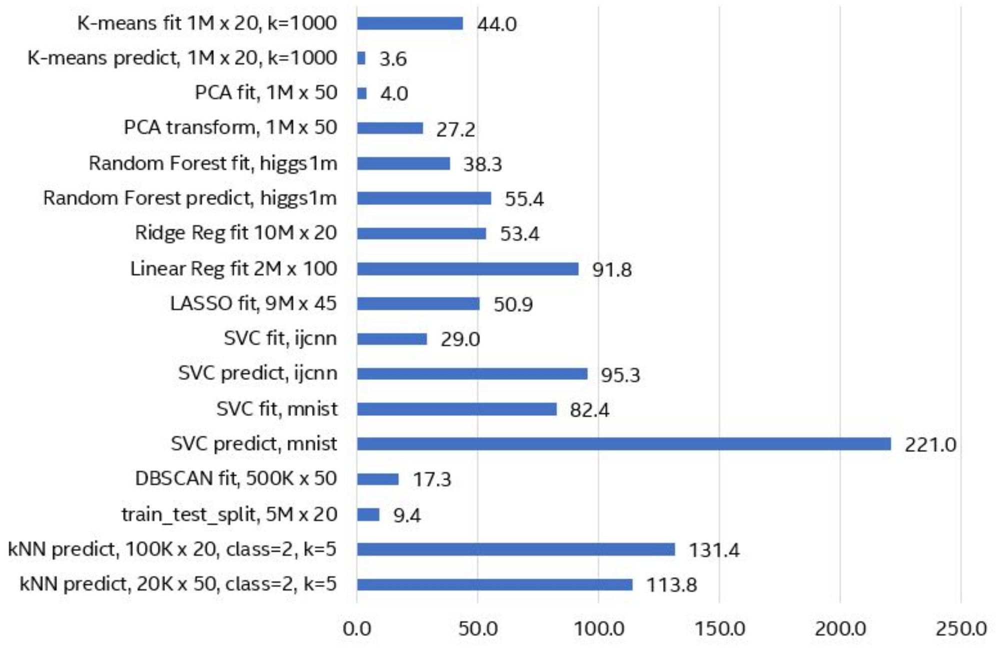
    - **Intel** CPU/GPU optimizations patching

```
from daal4py.sklearn import patch_sklearn
from daal4py.oneapi import sycl_context
patch_sklearn()
with sycl_context("gpu"):
```

  - Installation:

```
conda install daal4py -c intel
```

Speedups of daal4py-powered Scikit-learn over the original Scikit-learn



bit.ly/lskl_03

Key Takeaways

- Learning Scikit-learn means:
  - Understanding the machine learning workflows
  - Learning its API conventions
  - Familiarizing its documentations
  - Inspecting the example codes

- Making scikit-learn more productive means:
  - Making sure everything work fine without performance optimization
  - Carefully profiling the code to address the performance bottleneck
  - Considering the scikit-learn extensive libraries
  - Trying RAPIDS, daal4py, watching JAX