Python Applied to Machine Learning and Statistics

Lecture 03: scikit-learn

Pedro Costa Ricardo Cruz Kelwin Fernandes Adrián Galdrán
Chetak Kandaswamy

September 12, 2016

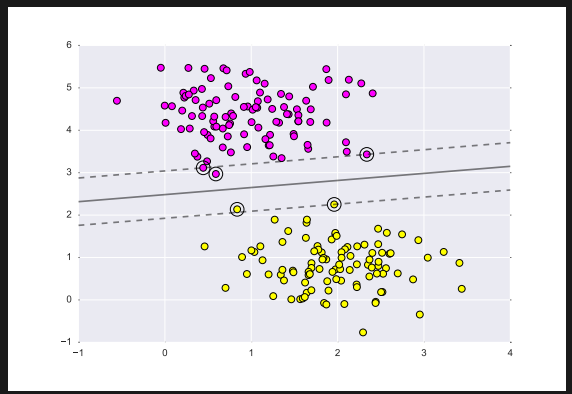
Machine Learning

- Subfield of Artificial Intelligence
- "Learn without being explicitly programmed" [1]
- Extract meaningful information from **data** and **generalize** to unseen examples

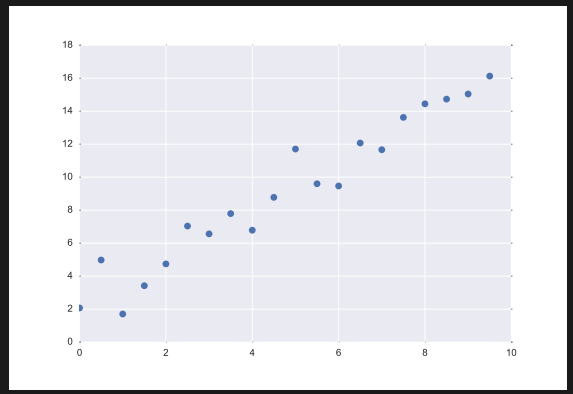
[1] Simon, Phil. Too Big to Ignore: The Business Case for Big Data. Vol. 72. John Wiley & Sons, 2013.

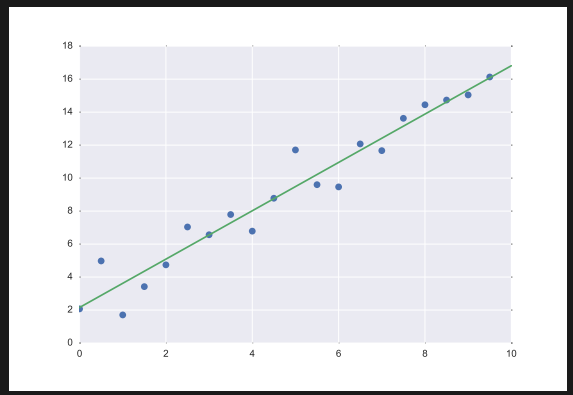
Machine Learning

Classification

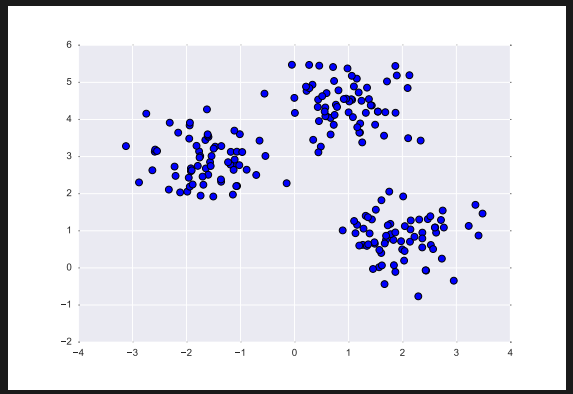


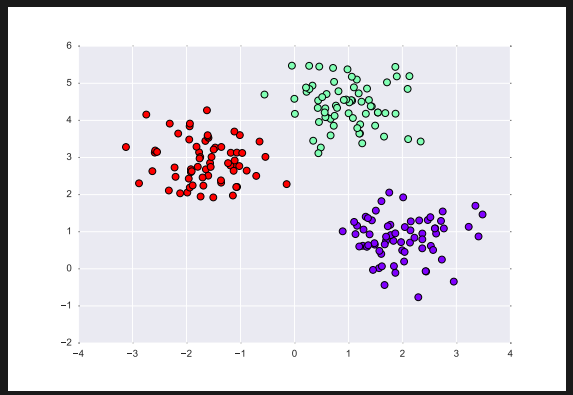
Regression





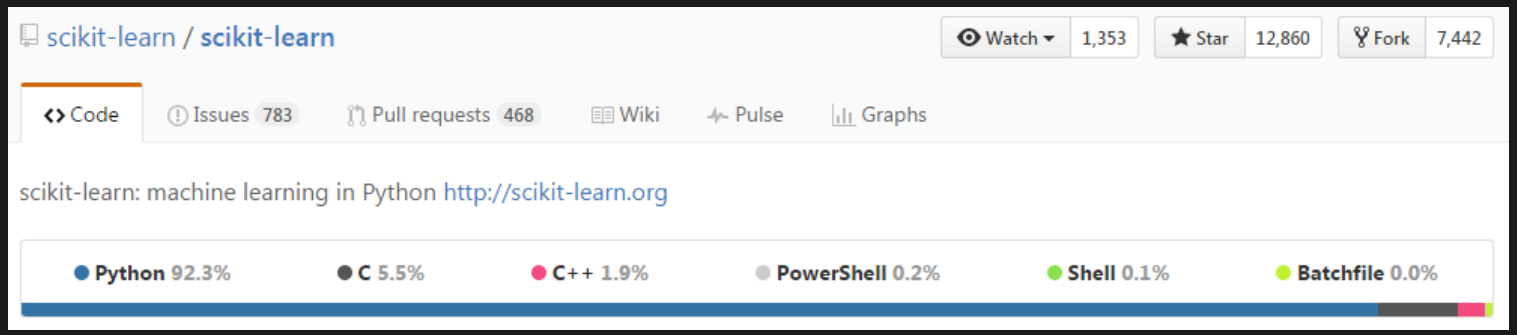
Clustering





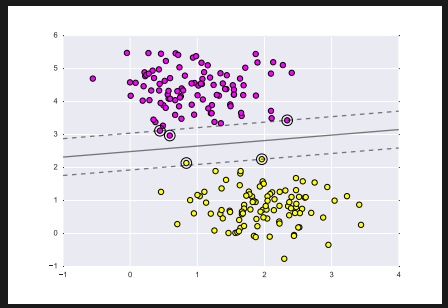
scikit-learn

- Python module for Machine Learning
- Built on top of NumPy, SciPy and Matplotlib
- Python 2.7 and Python 3.5
- Started in 2007 by David Cournapeau as a Google Summer of Code project
- Open-source

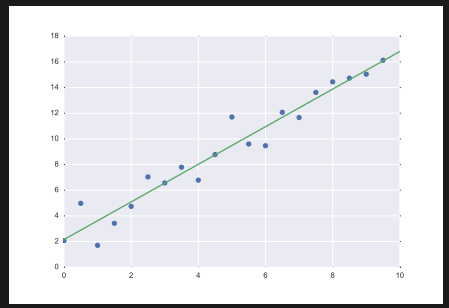


scikit-learn

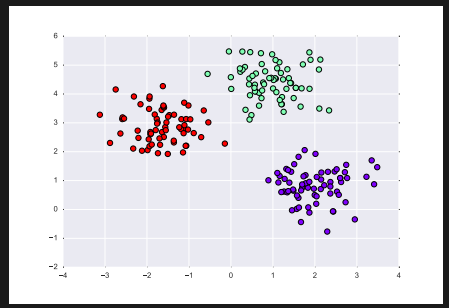
Classification



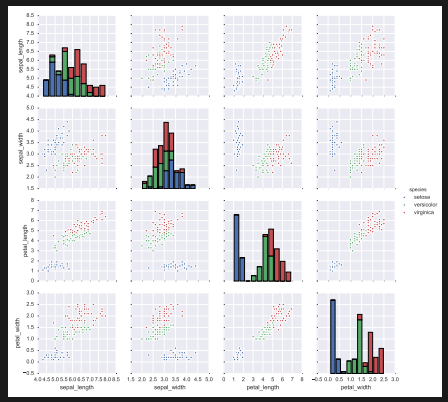
Regression



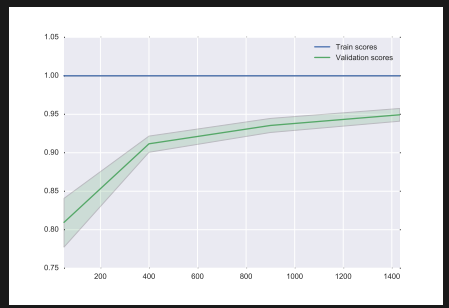
Clustering



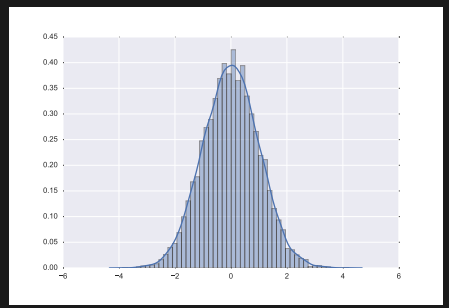
Dimensionality Reduction



Model Selection

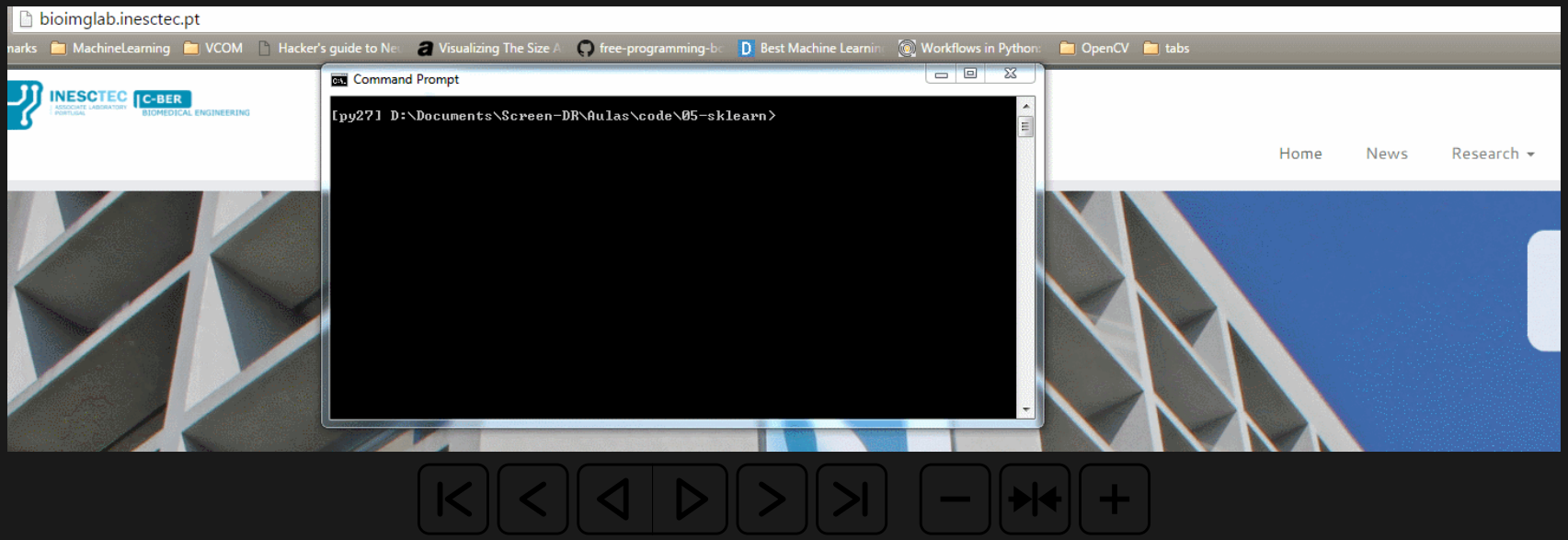


Preprocessing



Jupyter Notebook App

- *Jupyter Notebook App* is a server-client application;
- Runs *Notebook Documents* on a web browser;
- You can run it locally or on a remote server.



Notebook Document

- Documents that contain computer code and rich text elements;
- Human-readable: you can write a report to your supervisor with equations, tables and plots;
- Executable: you can write and execute code.

Kernel

- Launched when a notebook is opened;
- Computational engine that executes code;
- Started with the IPython kernel, but now there are kernels for dozens of other languages (i.e. IMatlab, IRKernel, C,...).

Agenda

Day 1
1. Classification
2. Regression
3. Clustering
4. Dimensionality Reduction

Day 2
1. Model Selection
2. Preprocessing

Objects

Estimator:	Implements the fit method to learn from data.
	For Supervised Learning:
	`estimator = estimatorObj.fit(data, labels)`
	and Unsupervised Learning:
	`estimator = estimatorObj.fit(data)`

Predictor:	`labels = predictorObj.predict(data)` May implement `predict_proba` to return the degree of certainty.
Transformer:	Filters or modifies the data: `new_data = transformerObj.transform(data)`
Model:	Measures goodness of fit: `score = modelObj.score(data, labels)`

Estimator

- The predominant object in scikit-learn;
- All estimators must inherit from `sklearn.base.BaseEstimator`;
- All the hyper-parameters are set at the initialization of the estimator:

```
model = LinearRegression(fit_intercept=True)
```

- The hyper-parameters' validity should be checked on the **fit** method;
- The model's parameters are learnt with the **fit** method;
- This means that the model is independent of the data.

This is wrong:

```
model = LinearRegression(X, y)
```

Fit

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

Parameters:

- X: an [n_samples x n_features] matrix.
  WARNING: Has to be a matrix even when n_features=1;
- y: an [n_samples] dimensional array.

In order to mix supervised and unsupervised estimators in a pipeline, even unsupervised estimators need to accept a y as argument.

Ugly, I know, but python makes it easier:

```
model.fit(X, y=None)
```

Fit

- It returns itself:

```
y_pred = LinearRegression().fit(X_train, y_train).predict(
    X_test)
```

- Fit is idempotent (unless the estimator has the *warm_start* parameter)
  - It "forgets" previously learned parameters;
  - Given the same data, always provides the same results.

Weights

- Some estimators allow to give different weights to different samples:

```
model.fit(X, y, sample_weight=None)
```

  - sample_weight is an [n_samples] float array.

- Other estimators allow to give different weights to different classes:

```
SVC(class_weight=None)
```

  class_weight can be:

  - *None* (default);
  - 'balanced' adjust weights inversely proportional to class frequencies;
  - {class: weight} dictionary that maps the class to the weight.

Parameters

The names of the estimator's parameters have a trailing underscore.

Examples:

- The coefficients of a linear regression estimator.

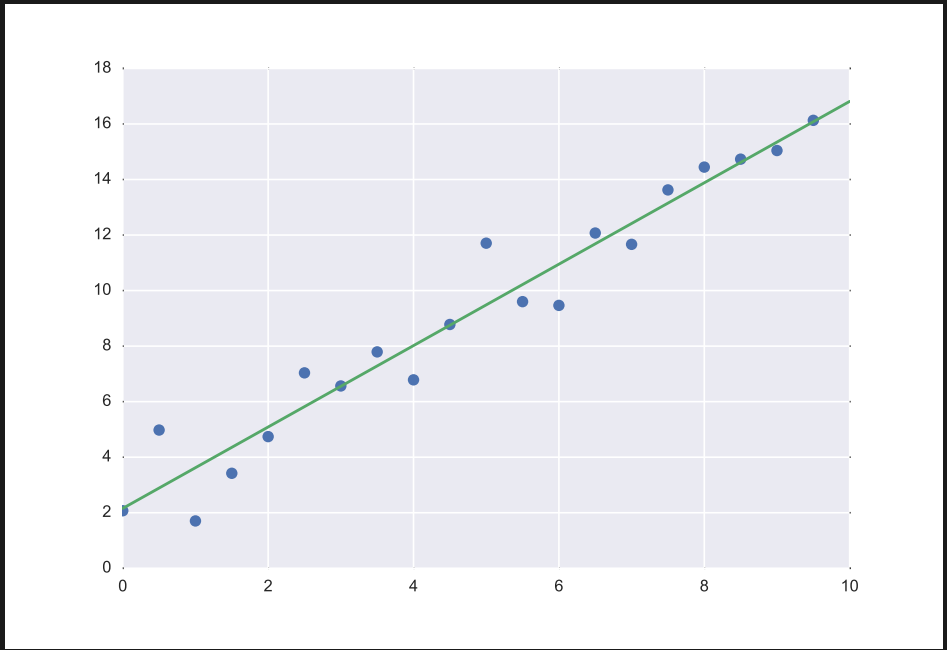
```
model.coef_
```

- The intercept term of a linear regression estimator.

```
model.intercept_
```

Estimator

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



(See notebook)

Datasets

3 main types of dataset generation:

1. Toy datasets (i.e. iris);
2. Sample images;
3. Sample generators.

Datasets

1. **Toy datasets (i.e. iris)**;
2. Sample images;
3. Sample generators.

- load_boston() - Regression;
- load_iris() - Classification;
- load_diabetes() - Regression;
- load_digits([n_class]) - Classification;
- load_linnerud() - Multivariate Regression.

Datasets

1. Toy datasets (i.e. iris);
2. **Sample images**;
3. Sample generators.

- load_sample_images() or load_sample_image(image_name):

china.jpg

flower.jpg

Datasets

1. Toy datasets (i.e. iris);
2. Sample images;
3. **Sample generators**.

For:

- Classification (i.e. make_blobs, make_classification, make_multilabel_classification,...);
- Clustering (i.e. make_biclusters, make_checkboard);
- Regression (i.e. make_regression, make_friedman1, make_sparse_uncorrelated,...);
- Manifold Learning (i.e. make_s_curve, make_swiss_roll);
- Decomposition (i.e. make_low_rank_matrix, make_spd_matrix,...)

Datasets

Others:

- Olivetti faces dataset;
- The 20 newsgroups text dataset;
- Datasets from mldata.org:

```
1 from sklearn.datasets import fetch_mldata
2 mnist = fetch_mldata('MNIST original')
```

Interface

- **Supervised:**
  - **model.predict(data)**: An array containing the classes of every example in *data*.
  - **model.predict_proba(data)**: The probability of belonging to each of the classes.
  - **model.score(data, labels)**: A measure of how good the fit was. Scores lie between 0 and 1.
- **Unsupervised:**
  - **model.predict(data)**: The cluster of each example.
  - **model.transform(data)**: Maps the data into the model's space.

Some estimator implement the **fit_predict** or **fit_transform** for efficiency reasons.

Your own Estimator

- Inherit from the *BaseEstimator* class which already takes care of the *get_params* and *set_params* functions.
- Inherit from one of the *Mixins* defined on the base module. These are the most important:
  - **ClassifierMixin**: defines the *score* function as the accuracy.
  - **RegressorMixin**: defines the *score* function as the R^2.
  - **ClusterMixin**: defines the *fit_predict* function.
- All these *Mixins* set the *_estimator_type* attribute that is used by some scikit-learn methods.

Your own Estimator

```python
class NearestNeighbor(BaseEstimator, ClassifierMixin):

  def __init__(self, demo_argument=True):
    # Parameters should have the same name as attibutes!
    self.demo_argument = demo_argument

  def fit(self, X, y):
    # Need to save classes_
    self.classes_ = unique_labels(y)

    # No actual fit. Just save the data.
    self.X_ = X
    self.y_ = y

    # Fit returns itself.
    return self

  def predict(self, X):
    closest = np.argmin(euclidean_distances(X, self.X_), axis=1)
    return self.y_[closest]
```

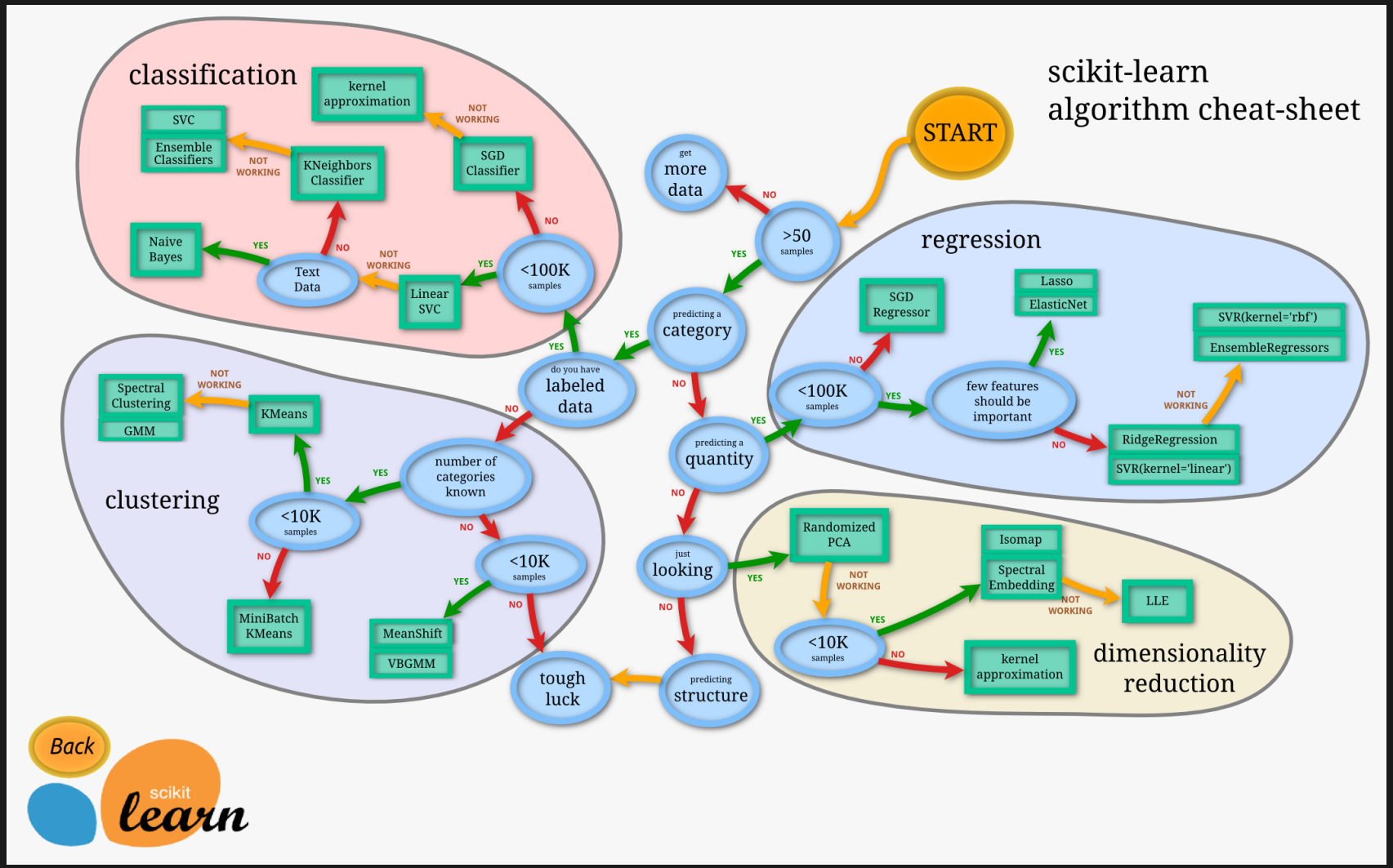
Your own Estimator

- It is a good practice to validate the arguments of the functions. Scikit-learn provides some methods to do that in the *sklearn.utils.validation* module such as:
  - **check_X_y**: Checks if X and y have valid dimensions.
  - **check_array** : Checks if a variable is a valid array and, if not, converts it into one.
  - **check_is_fitted**: Checks if a list of variables are set.
- You should also run your estimator through the *check_estimator* method. This checks whether the estimator is compatible with the scikit-learn interface or not.

Final Notes

- There is a python library named *xgboost* which implements Gradient Boosted Trees that consistently win Kaggle competitions. We are not going to cover that, but the API is similar to scikit-learn's.
- What about Neural Networks? The new version will support supervised neural networks for classification and regression, but:
  - Do not run on GPU;
  - Low flexibility. There is no Dropout and no support for more complex architectures such as ResNets;
  - No Recurrent Neural Networks;
  - ...
- In summary, only use them for very simple and very small problems. We will cover better tools to work with Neural Networks in following classes.

Flow Chart


scikit-learn
algorithm cheat-sheet
START
get more data
>50 samples
predicting a category
do you have labeled data
predicting a quantity
just looking
predicting structure
tough luck
classification
kernel approximation
SVC
Ensemble Classifiers
KNeighbors Classifier
SGD Classifier
Naive Bayes
Text Data
Linear SVC
<100K samples
regression
SGD Regressor
Lasso
ElasticNet
SVR(kernel='rbf')
EnsembleRegressors
<100K samples
few features should be important
RidgeRegression
SVR(kernel='linear')
clustering
Spectral Clustering
GMM
KMeans
number of categories known
<10K samples
MiniBatch KMeans
<10K samples
MeanShift
VBGMM
dimensionality reduction
Randomized PCA
Isomap
Spectral Embedding
LLE
<10K samples
kernel approximation
YES
NO
NOT WORKING
Back
scikit learn