SkLearn

Stéphane Canu
stephane.canu@insa-rouen.fr

Advanced Machine Learning

Winter 2023-2024

Scikit-learn: a library to do Machine Learning in Python

- Scikit-learn is a free software machine learning library in Python. (also known as sklearn)
- It is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.
- In 2010 Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort and Vincent Michel (from Inria, France) took leadership of the project and made the first public release (February 1, 2010).

scikit-learn

Machine Learning in Python

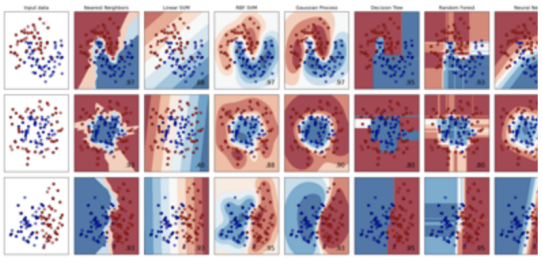
Getting Started | Release Highlights for 1.0 | GitHub

- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying which category an object belongs to.

Applications: Spam detection, image recognition.
Algorithms: SVM, nearest neighbors, random forest, and more...

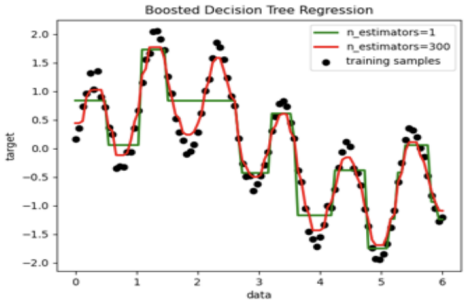


Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: SVR, nearest neighbors, random forest, and more...

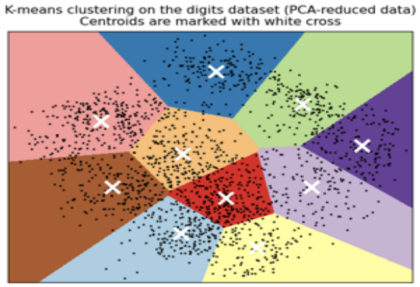


Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, and more...

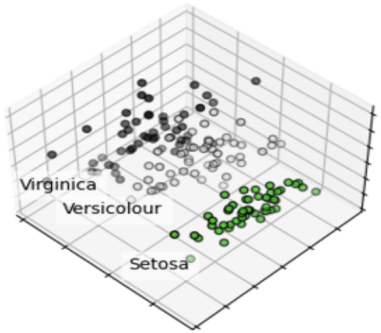


Examples

Dimensionality reduction

Reducing the number of random variables to consider.

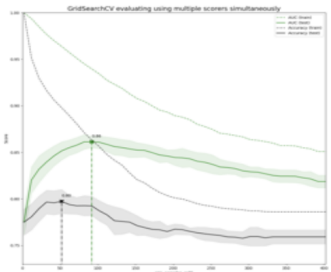
Applications: Visualization, Increased efficiency
Algorithms: k-Means, feature selection, non-negative matrix factorization, and more...



Model selection

Comparing, validating and choosing parameters and models.

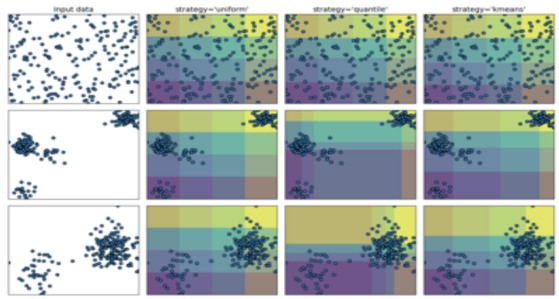
Applications: Improved accuracy via parameter tuning
Algorithms: grid search, cross validation, metrics, and more...



Preprocessing

Feature extraction and normalization.

Applications: Transforming input data such as text for use with machine learning algorithms.
Algorithms: preprocessing, feature extraction, and more...



Le Github de scikit-learn

COPYING	MNT Update license year to 2022 (#24418)	2 months ago
MANIFEST.in	BLD Migrate away from distutils and only use setuptools (#24563)	21 hours ago
Makefile	MAINT simplify linting by running flake8 on the whole project (#238...	4 months ago
README.rst	MAINT Bump min dependencies for 1.2 (#24650)	21 days ago
SECURITY.md	DOC Update SECURITY.md for 1.1.3 (#24783)	4 days ago
azure-pipelines.yml	BLD Migrate away from distutils and only use setuptools (#24563)	21 hours ago
conftest.py	MAINT Move conftest up a level (#20208)	17 months ago
pyproject.toml	MAINT force NumPy version for building scikit-learn for CPython 3.1...	10 days ago
setup.cfg	MAINT Introduce `MiddleTermComputer`, an abstraction generalizing ...	20 hours ago
setup.py	MAINT Introduce `MiddleTermComputer`, an abstraction generalizing ...	20 hours ago

README.rst

Azure Pipelines succeeded | build passing | codecov 97% | circleci passing | Wheel builder failing | code style black

python 3.8 | 3.9 | 3.10 | pypi v1.1.3 | DOI 10.5281/zenodo.7254371 | Benchmarked by asv

scikit-learn is a Python module for machine learning built on top of SciPy and is distributed under the 3-Clause BSD license.

The project was started in 2007 by David Cournapeau as a Google Summer of Code project, and since then many volunteers have contributed. See the About us page for a list of core contributors.

It is currently maintained by a team of volunteers.

Website: https://scikit-learn.org

Used by 398k

+ 398,452

Contributors 2,520

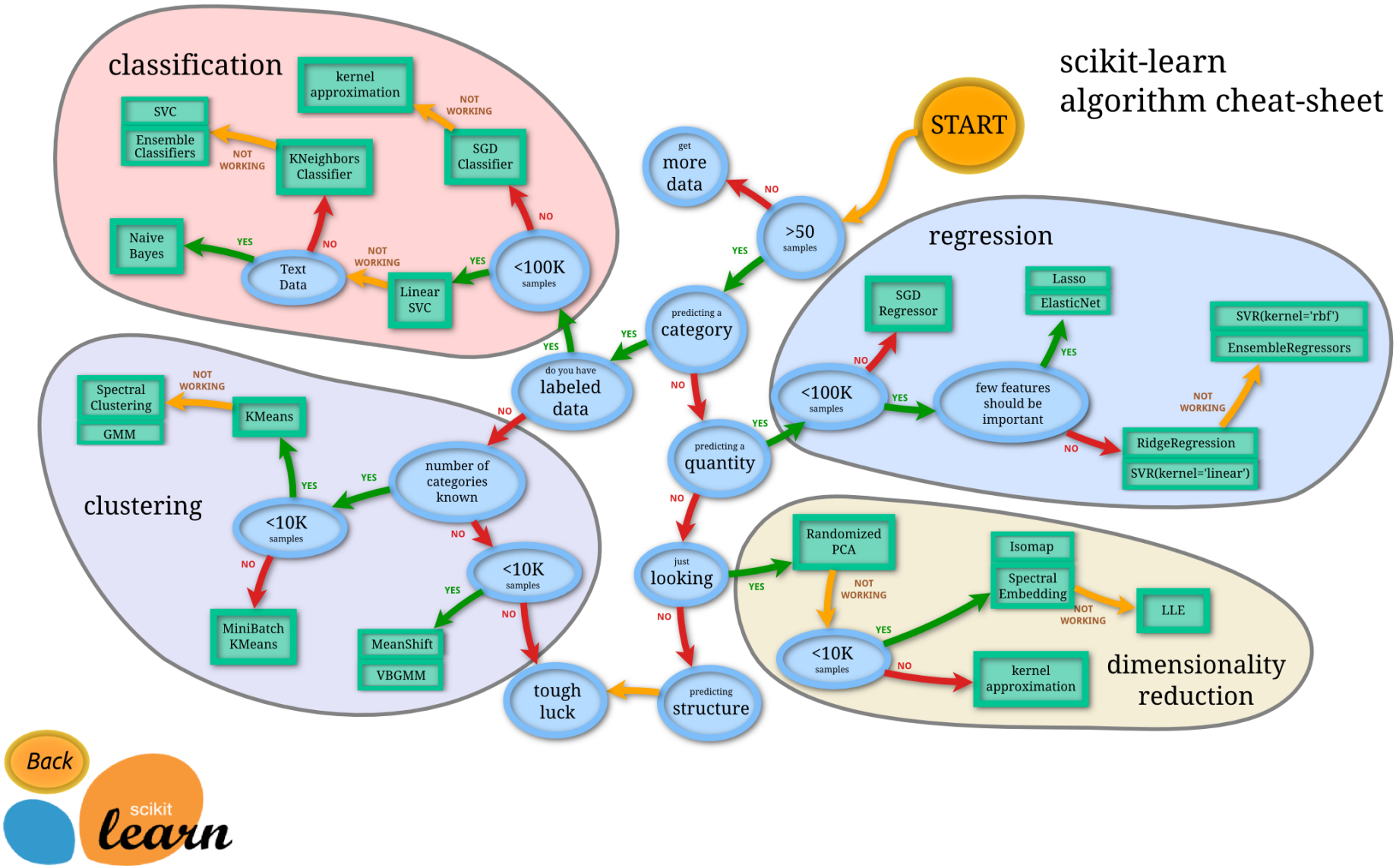
+ 2,509 contributors

Languages

Python 92.0% Cython 5.9%
C++ 1.3% Other 0.8%

https://github.com/scikit-learn/scikit-learn

Les méthodes de sklearn



A proper Scikit Learn Tutorial

https://github.com/jakevdp/sklearn_tutorial

https://github.com/jakevdp/sklearn_tutorial

e Lefeuvre 2022.ecmlpkdd FoE Overleaf ECML genome_covid PRONOTE Adele Drive cantine Galaxie_CS Commite_IA DARI retraite Gael

README.md

Scikit-learn Tutorial

Jake VanderPlas

- email: jakevdp@uw.edu
- twitter: @jakevdp
- github: jakevdp

This repository contains notebooks and other files associated with my Scikit-learn tutorial.

Installation Notes

This tutorial requires the following packages:

- Python version 2.6-2.7 or 3.3+
- `numpy` version 1.5 or later: http://www.numpy.org/
- `scipy` version 0.10 or later: http://www.scipy.org/
- `matplotlib` version 1.3 or later: http://matplotlib.org/
- `scikit-learn` version 0.14 or later: http://scikit-learn.org
- `ipython` version 2.0 or later, with notebook support: http://ipython.org
- `seaborn` version 0.5 or later

Packages

No packages published

Contributors 3

jakevdp Jake Vanderplas

arokem Ariel Rokem

conkerts

Languages

Jupyter Notebook 81.8%

Python 18.2%

Scikit-Learn Tutorial

Jake VanderPlas

This is the main index for the materials of my Scikit-Learn tutorial. Please refer to the github repository for this tutorial for any updates.

Tutorial Notebooks

The following links are to notebooks containing the tutorial materials. Note that many of these require files that are in the directory structure of the github repository in which they are contained. There is not time during the tutorial to cover all of this material, but I left it in in case attendees would like to go deeper on their own.

1. Preliminaries

- 01-Preliminaries.ipynb

2. Introduction to Machine Learning with Scikit-Learn

- 02.1-Machine-Learning-Intro.ipynb
- 02.2-Basic-Principles.ipynb

3. Supervised Learning In-Depth

- 03.1-Classification-SVMs.ipynb
- 03.2-Regression-Forests.ipynb

4. Unsupervised Learning In-Depth

- 04.1-Dimensionality-PCA.ipynb
- 04.2-Clustering-KMeans.ipynb
- 04.3-Density-GMM.ipynb

5. Model Validation In-Depth

- 05-Validation.ipynb

Data in Scikit Learn

- 7 Toy, 7 Real world, 4 Generated and 4 other datasets
- Loading other datasets
  - Sample images
  - Datasets in svmlight / libsvm format
  - Downloading datasets from the openml.org repository

In [ ]:
```
from sklearn.datasets import load_iris
iris = load_iris()
```

Load CSV files using panda

```
import pandas as pd

in_file = 'spaceship_titanic_train.csv'
data_frame = pd.read_csv(in_file)
data_frame.head()
```

	PassengerId	HomePlanet	CryoSleep	Cabin	Destination	Age	VIP	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck	Name	Transported
0	0001_01	Europa	False	B/0/P	TRAPPIST-1e	39.0	False	0.0	0.0	0.0	0.0	0.0	Maham Ofracculy	False
1	0002_01	Earth	False	F/0/S	TRAPPIST-1e	24.0	False	109.0	9.0	25.0	549.0	44.0	Juanna Vines	True
2	0003_01	Europa	False	A/0/S	TRAPPIST-1e	58.0	True	43.0	3576.0	0.0	6715.0	49.0	Altark Susent	False
3	0003_02	Europa	False	A/0/S	TRAPPIST-1e	33.0	False	0.0	1283.0	371.0	3329.0	193.0	Solam Susent	False
4	0004_01	Earth	False	F/1/S	TRAPPIST-1e	16.0	False	303.0	70.0	151.0	565.0	2.0	Willy Santantines	True

Data pre processing: encode multiclass label

```
df=df.drop('id',axis=1)
```

```
df['prognosis'].value_counts()

from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
df['prognosis'] = label_encoder.fit_transform(df['prognosis'])
```

```
df['prognosis'].value_counts()
Out[2]: West_Nile_fever          85
        Japanese_encephalitis    81
        Rift_Valley_fever        70
        Tungiasis                70
        Chikungunya              66
        Dengue                   63
        Yellow_Fever             61
        Zika                     58
        Plague                   53
        Lyme_disease             52
        Malaria                  48
        Name: prognosis, dtype: int64
```

intial labels

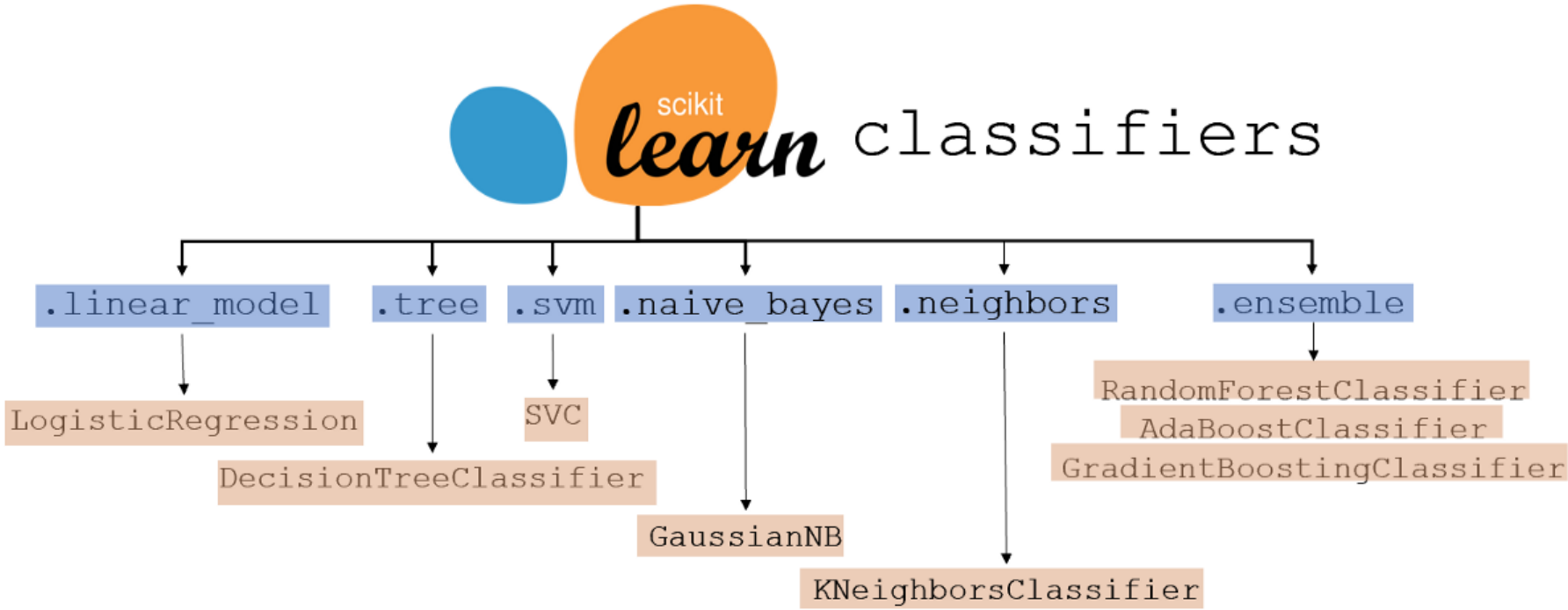
```
[5]: print(df['prognosis'])
0       3
1       7
2       3
3      10
4       6
      ..
702     5
703     4
704    10
705     5
706     7
Name: prognosis, Length: 707, dtyp
```

class label

```
rée [24]: from sklearn.preprocessing import OneHotEncoder

OneHot_prognosis = df['prognosis'].values.reshape(-1, 1)
enc = OneHotEncoder().fit(OneHot_prognosis)
OneHot_prognosis = enc.transform(OneHot_prognosis).toarray()
for i in range (5):
    print(*OneHot_prognosis[i][:])
OneHot_prognosis.shape

0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0
0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0
0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0
Out[24]: (707, 11)
```

one hot encoding

sklearn classifier models




```
#import
from sklearn.branch import model_name

#create instance
model = model_name()

#fit model
model.fit(X_train, y_train)
```

A typical classification process

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_a, y_a, test_size=0.25, random_state=42)
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from sklearn import metrics

model_dt = DecisionTreeClassifier()
model_dt.fit(X_train, y_train)
y_pred = model_dt.predict(X_test)

cf_dt = confusion_matrix(y_test, y_pred)*100/len(y_test)
print('Basis: Decision tree')
print(cf_dt)
print(metrics.classification_report(y_test, y_pred, target_names=['Drowned', 'Survivor']))
print('the error rate is: %5.2f %%' % (cf_dt[0,1]+cf_dt[1,0]))
```

```
Basis: Decision tree
[[33.34866605 16.42134315]
 [11.08555658 39.14443422]]
              precision    recall  f1-score   support

     Drowned       0.75      0.67      0.71      1082
    Survivor       0.70      0.78      0.74      1092

    accuracy                           0.72      2174
   macro avg       0.73      0.72      0.72      2174
weighted avg       0.73      0.72      0.72      2174

the error rate is: 27.51 %
```

Multiclass evaluation: Mean Average Precision MAP@k

Average Precision AP@k

$$\underset{t}{\text{target}} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad \underset{\text{p}}{\text{proba}} = \begin{pmatrix} 0.12 \\ 0.32 \\ 0.31 \\ 0.09 \\ 0.16 \\ 0.02 \end{pmatrix} \quad \underset{r}{\text{rank}} = \begin{pmatrix} 4 \\ 1 \\ 2 \\ 5 \\ 3 \\ 6 \end{pmatrix} \quad \underset{s}{\text{sort}} = \begin{pmatrix} 2 \\ 3 \\ 5 \\ 1 \\ 4 \\ 6 \end{pmatrix}$$

$$AP@k = \sum_{\kappa=1}^{k} \frac{\mathbb{1}_{\{t(s(\kappa))=1\}}}{\kappa}$$

Mean over all examples

Prediction post processing (1)

```
# Lets take a look at getting our top 3 prognoses for just a single prediction first
print("Output of predict_proba:")
print(predictions[0])

# We can get the indices of the highest probabilities with argsort
sorted_prediction_ids = np.argsort(-predictions[0]) # Note argsort sorts in ascending order,
print("Indices sorted by probabilities:")
print(sorted_prediction_ids)
# 2 is our first id, and the probability at index 2 from predictions[0] is the highest
```

```
Output of predict_proba:
[0.33 0.15 0.22 0.   0.03 0.02 0.08 0.04 0.03 0.07 0.03]
Indices sorted by probabilities:
[ 0  2  1  6  9  7  4  8 10  5  3]
```

Prediction post processing (2)

[28]:
```
# Lets take a look at getting our top 3 prognoses (inverse decision score):
sorted_prediction_ids = np.argsort(-predictions, axis=1)
top_3_prediction_ids = sorted_prediction_ids[:,:3]
top_3_prediction_ids[:10] # Spot check our first 10 values
```

:[28]:
```
array([[ 6,  2,  7],
       [ 4,  1,  5],
       [ 8,  2,  1],
       [ 7,  8,  6],
       [ 4,  2,  3],
       [ 8, 10,  5],
       [ 4,  9,  2],
       [ 0,  1,  9],
       [ 9,  2, 10],
       [ 8,  9,  2]])
```

[29]:
```
# Because enc.inverse_transform expects a specific shape (a 2D array with 1 column) we can save the original shape t
original_shape = top_3_prediction_ids.shape
top_3_predictions = label_encoder.inverse_transform(top_3_prediction_ids.reshape(-1, 1))
top_3_predictions = top_3_predictions.reshape(original_shape)
top_3_predictions[:10] # Spot check our first 10 values
```

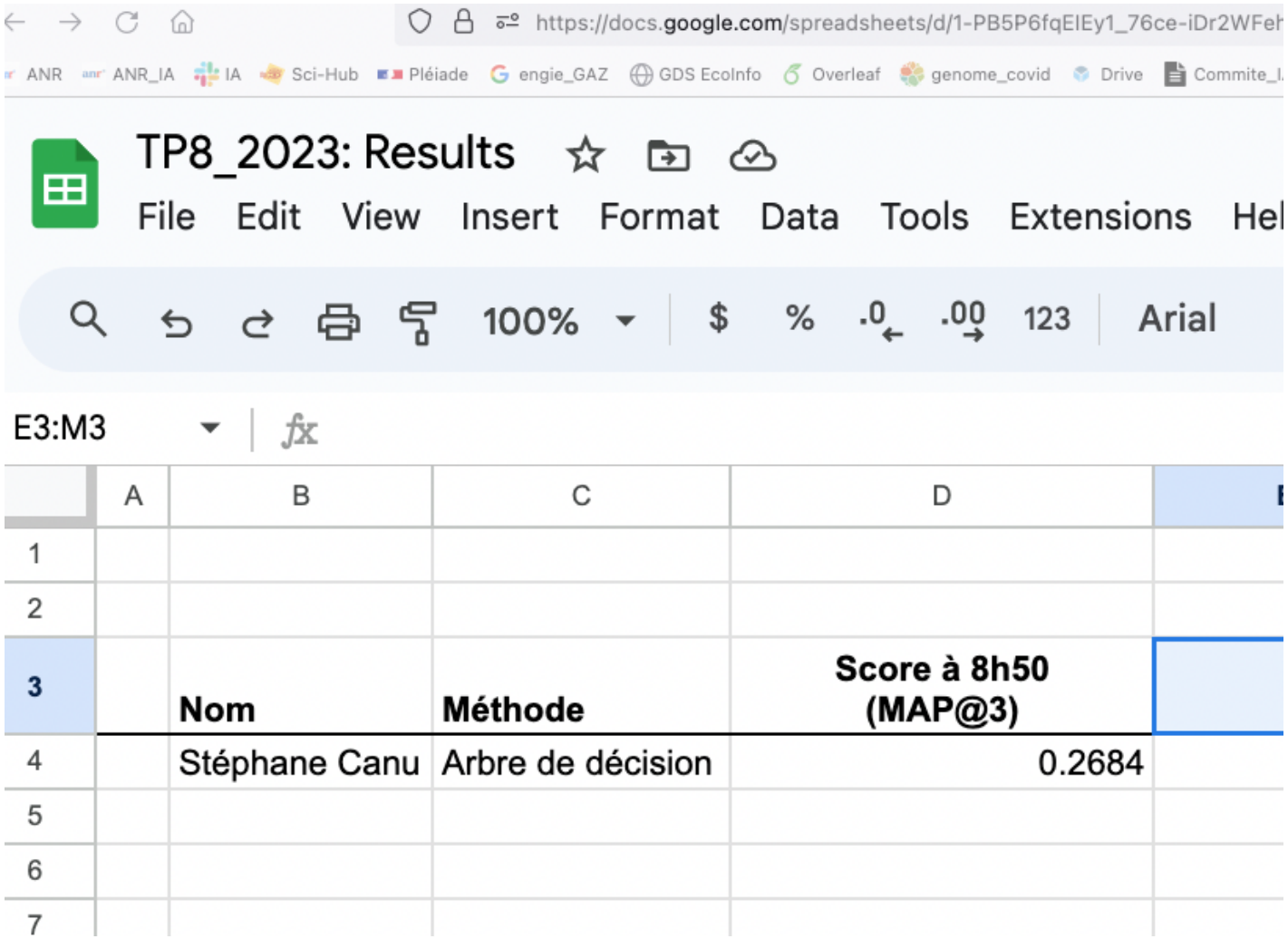
:[29]:
```
array([['Rift_Valley_fever', 'Japanese_encephalitis', 'Tungiasis'],
       ['Malaria', 'Dengue', 'Plague'],
       ['West_Nile_fever', 'Japanese_encephalitis', 'Dengue'],
       ['Tungiasis', 'West_Nile_fever', 'Rift_Valley_fever'],
       ['Malaria', 'Japanese_encephalitis', 'Lyme_disease'],
       ['West_Nile_fever', 'Zika', 'Plague'],
       ['Malaria', 'Yellow_Fever', 'Japanese_encephalitis'],
       ['Chikungunya', 'Dengue', 'Yellow_Fever'],
       ['Yellow_Fever', 'Japanese_encephalitis', 'Zika'],
       ['West_Nile_fever', 'Yellow_Fever', 'Japanese_encephalitis']],
      dtype=object)
```

Compute the Menb Average Precision MAP@k=3

```
# Models are evaluated based on MPA@3
mapk(y_test.values.reshape(-1, 1), top_3_prediction_ids, k=3)
```

Now improve on it using sklearn...

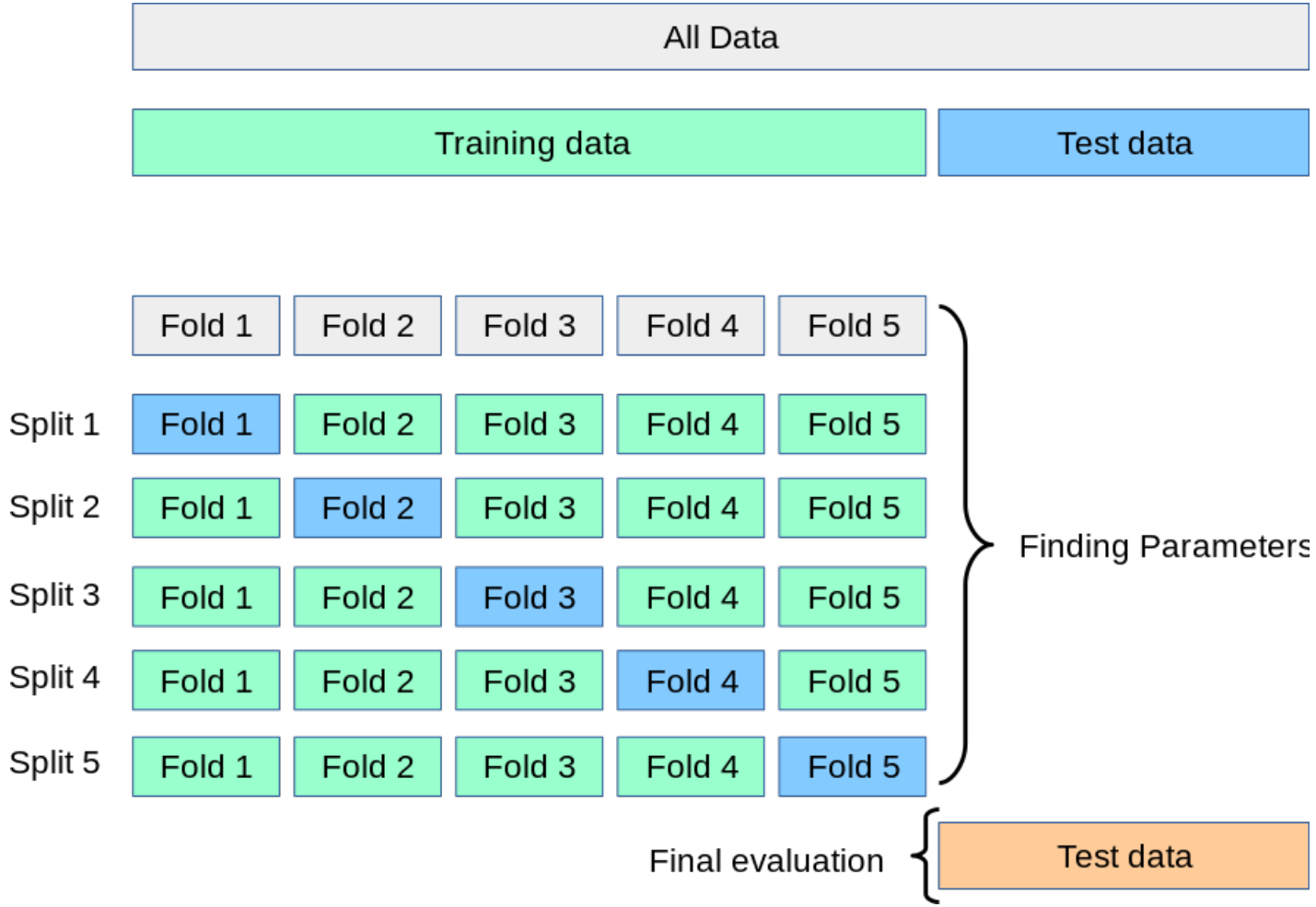
- choose your favorite method and train it with the train dataset
- report your result on the google doc from moodle BEFORE 8:50 today



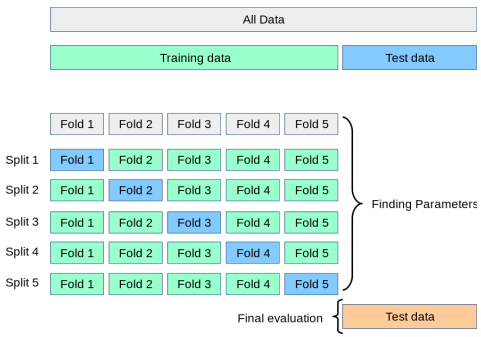
Advanced sklearn features

- Grid search cross validation
- Pipeline
- AutoSklearn

Grid search cross validation



Grid search cross validation

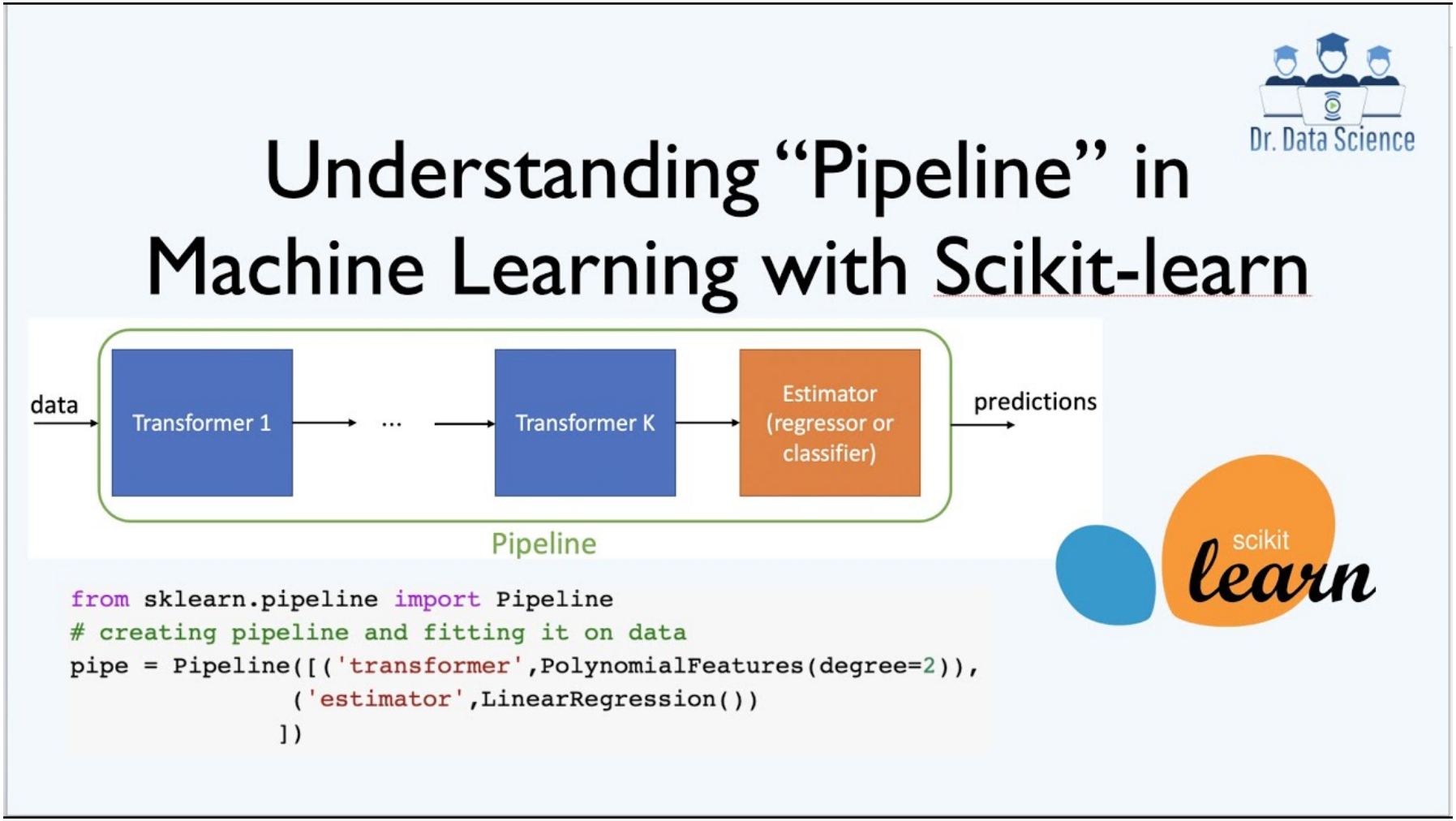



```
from sklearn.tree import DecisionTreeClassifier
from pprint import pprint

model_dt = DecisionTreeClassifier()
pprint(model_dt.get_params())
```

```
{'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'random_state': None,
 'splitter': 'best'}
```

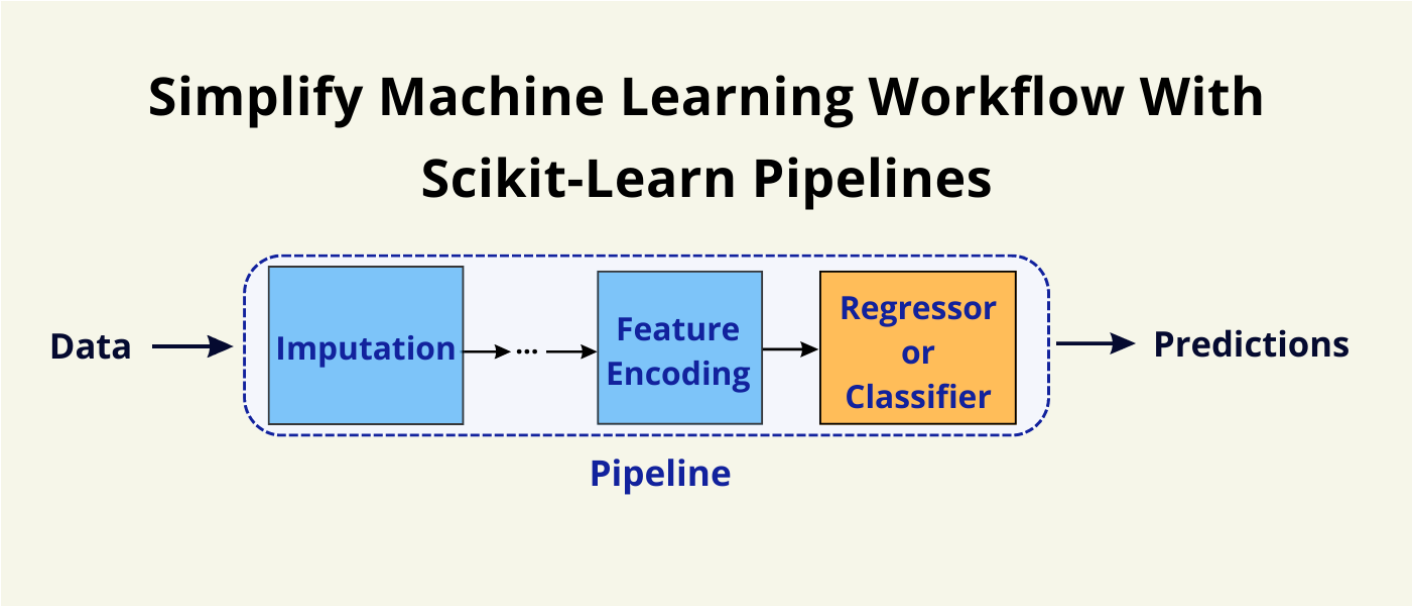
```
from sklearn.model_selection import GridSearchCV

param_grid = {'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
              'min_samples_leaf': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
              'max_features': [8, 9, 10, 11, 12],

              }
model_dt_cv = GridSearchCV(DecisionTreeClassifier(), param_grid, cv=10,
                           n_jobs = -1,
                           verbose = 2)
model_dt_cv.fit(X_train, y_train)
print(model_dt_cv.best_params_)

y_pred = model_dt_cv.predict(X_test)
```

Pipeline



The major advantage of using pipelines is we can optimize the entire training workflow, not just the learning algorithm.

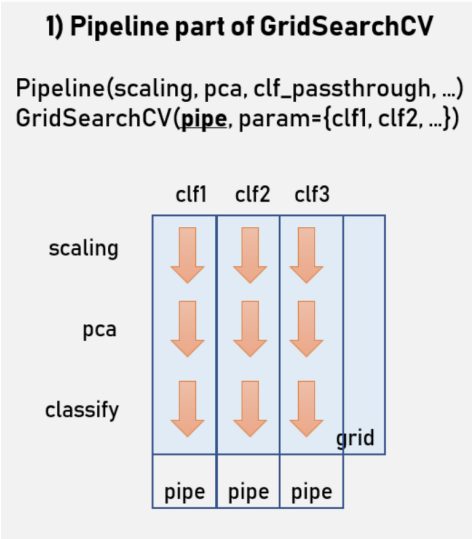
Pipeline example



Pipeline exemple

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline

scaler = StandardScaler()
pca = PCA()
logistic = LogisticRegression(max_iter=10000, tol=0.1)

pipe = Pipeline(steps=[("scaler", scaler), ("pca", pca), ("logistic", logistic)])
pipe.fit(X_train, y_train)

y_pred = pipe.predict(X_test)
```

Pipeline + GridSearchCV




```
scaler = StandardScaler()
pca = PCA()
logistic = LogisticRegression(max_iter=10000, tol=0.1)

pipe = Pipeline(steps=[("scaler", scaler), ("pca", pca), ("logistic", logistic)])

param_grid = {
    "pca__n_components": [4, 6, 8, 10, 12],
    "logistic__C": np.logspace(-4, 4, 6),
}
search = GridSearchCV(pipe, param_grid, n_jobs=2)
search.fit(X_train, y_train)

print(search.best_params_)

y_pred = search.predict(X_test)
cf_lr = confusion_matrix(y_test, y_pred)*100/len(y_test)
print('the error rate is: %5.2f %%' % (cf_lr[0,1]+cf_lr[1,0]))
```

```
{'logistic__C': 6.309573444801943, 'pca__n_components': 10}
the error rate is: 23.18 %
```

Conclusion

See Common pitfalls and recommended practices
https://scikit-learn.org/stable/common_pitfalls.html#common-pitfalls-and-recommended-practices