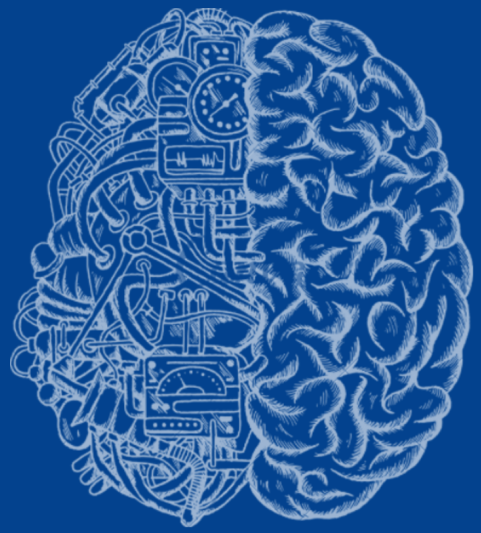
AARHUS
UNIVERSITY
SCHOOL OF ENGINEERING

MACHINE LEARNING

LESSON 3: Classification

CARSTEN EIE FRIGAARD

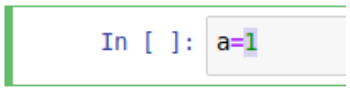
SPRING 2019



RESUMÉ: Jupyter Crash Course

Jupyter shortcuts:

- To modes: command mode (blue) and edit-mode (green),



- ESC: goto command mode (from edit mode),

Keyboard shortcuts

The Jupyter Notebook has two different keyboard input modes. **Edit mode** allows you to type code/text into a cell and is indicated by a green cell border. **Command mode** binds the keyboard to notebook level actions and is indicated by a grey cell border with a blue left margin.

Command Mode (press `Esc` to enable)

`F`: find and replace
`Ctrl-Shift-P`: open the command palette
`Enter`: enter edit mode
`Shift-Enter`: run cell, select below
`Ctrl-Enter`: run selected cells
`Alt-Enter`: run cell, insert below

`Shift-J`: extend selected cells below
`A`: insert cell above
`B`: insert cell below
`X`: cut selected cells
`C`: copy selected cells
`Shift-V`: paste cells above

RESUMÉ: Python Libraries Crash Course

A lot of modules/libraries are available for python, here we will use:

- `numpy`: numerical data representation module, for say vectors, matrices etc,
- `matplotlib`: Matplotlib is a Python 2D plotting library which produces publication quality figures.

Other libraries, typically used in ML, are:

- `pandas`: python data analysis library, a module for loading/saving and handling large data set,
- `scipy`: python library used for scientific computing and technical computing.

but we try to stick to `numpy` *in this course,*
...and note that `numpy.matrix` *is depricated!*

RESUMÉ: Matplotlib Crash Course

Visualizations can be created in multiple ways:

- `matplotlib`
- `pandas`: (via matplotlib),
- `seaborn`: statistically-focused plotting methods.

And we will stick to `matplotlib`, don't re-invent the wheel; find demos here

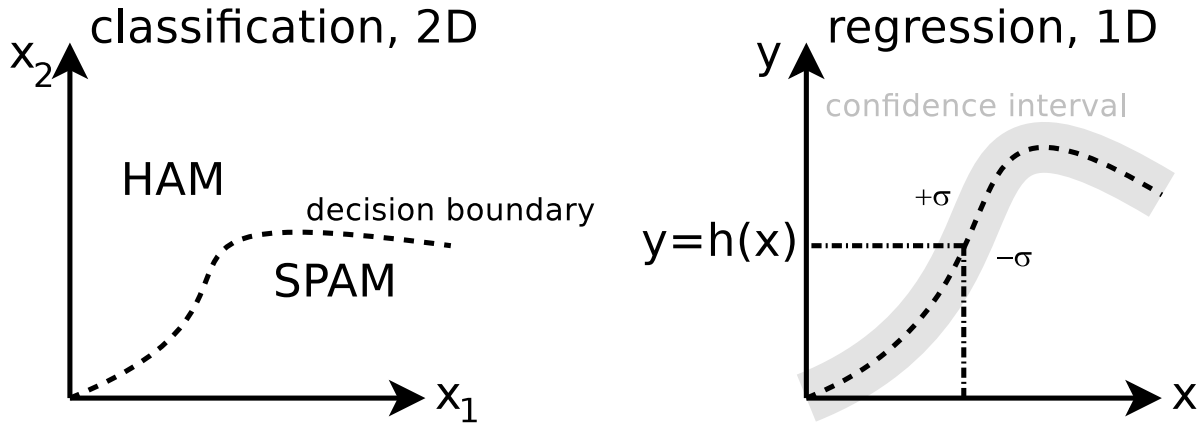
`https://matplotlib.org/gallery/index.html`

Classification vs. Regression

Given the following

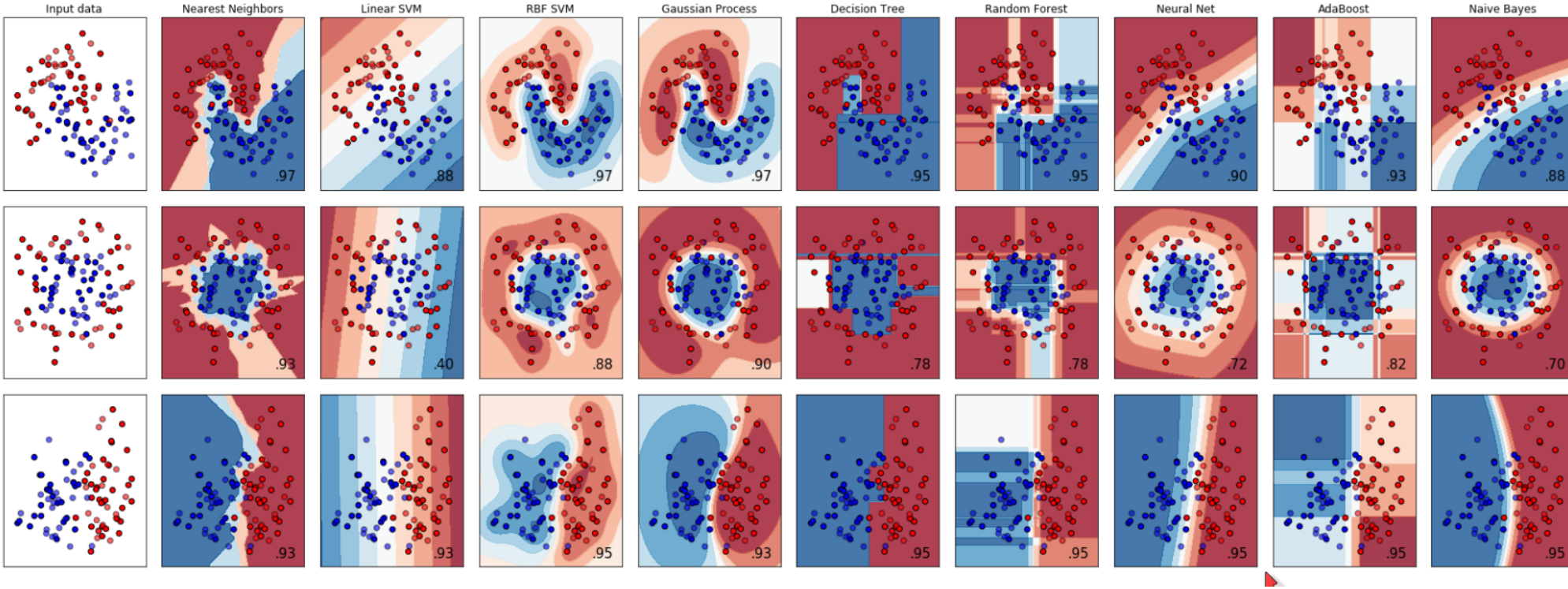
$$h : \mathbf{x} \rightarrow y$$

- if y is discrete/categorical variable, then this is **classification** problem.
- if y is real number/continuous, then this is a **regression** problem.



Classification

Decision Boundary for different Models



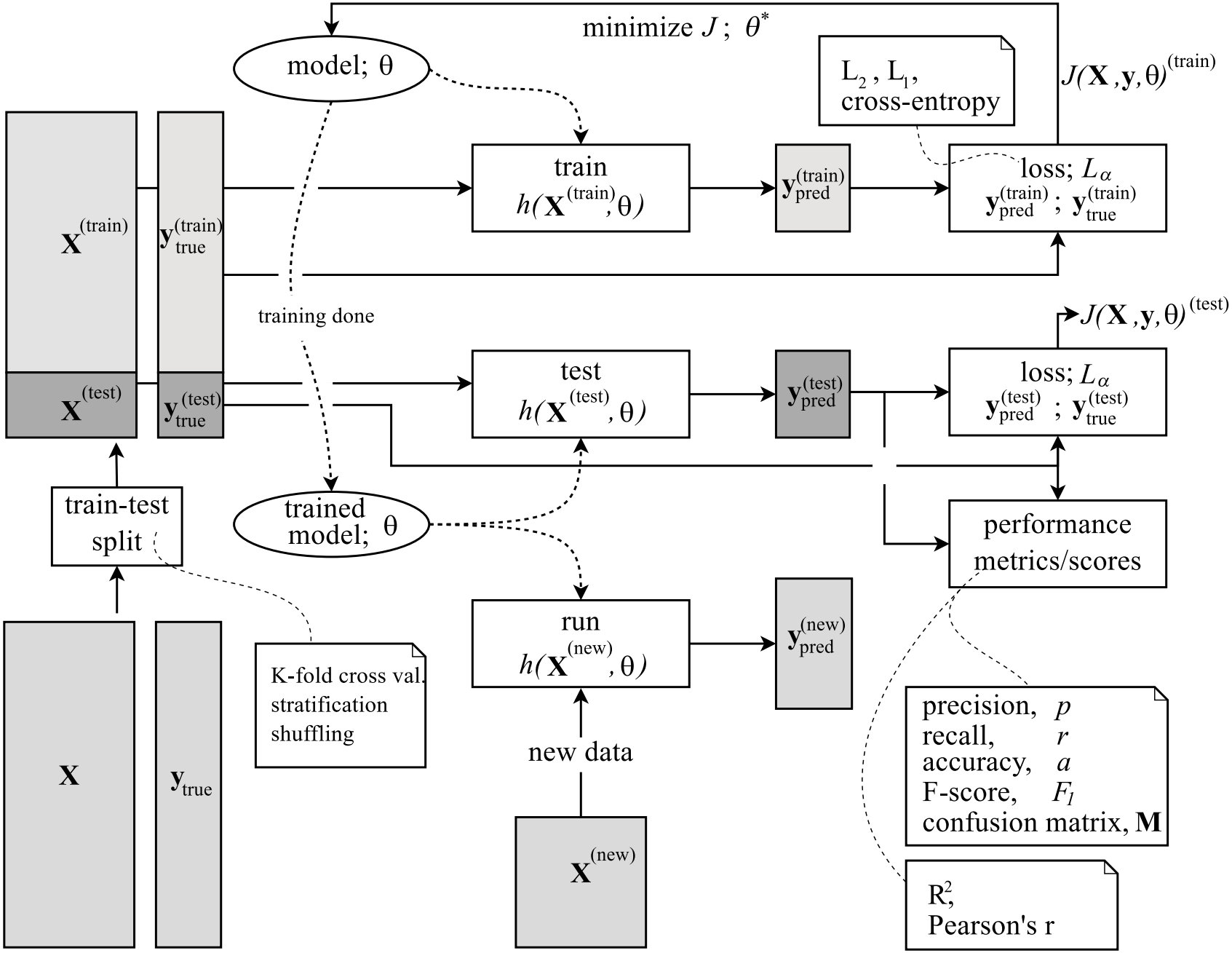
Souce code: `L03/Extra/plot_classifier_comparison.ipynb` in [GITMAL].

Multiclass Classification

Or Multinomial Classification
Introduction to Multilabel Classification

- Many classifiers are binary (HAM/SPAM)
- What to do for say a three category, like
  CAT/DOG/TURTLE
  problem?
- Divide into three CAT/NON-CAT, etc, binary classifiers and solve!
- Aka.: one-vs-rest (OvA),
  one-vs-all (OvA),
  one-against-all (OAA).
- Or the one-vs-one (OvO) method.
- NOTE: Multilabel classification is yet again different; it can categorize item into more classes, say both CAT and DOG!
- ...and Multioutput multiclass classification.

ML Supervised Learning



Exercise: L03/modules_and_classes.ipynb

Modules and Packages...

Python Basics ¶

Modules and Packages in Python

Reuse of code in Jupyter notebooks can be done by either including a raw python source as a magic command

```
%load filename.py
```

but this just pastes in the source and creates all kinds of pains regarding code maintenance.

A better way is to use a python **module**. A module consists simply (and pythonic) of a directory with a module init file in it (possibly empty)

```
libitmal/__init__.py
```

To this directory you can add modules in form of plain python files, say

```
libitmal/utils.py
```

That's about it! The `libitmal` file tree should now look like

```
libitmal/
├── __init__.py
├── __pycache__
│   ├── __init__.cpython-36.pyc
│   └── utils.cpython-36.pyc
├── utils.py
```

Exercise: L03/modules_and_classes.ipynb

Python classes...

Classes in Python

Good news: Python got classes. Bad news: they are somewhat obscure compared to C++ classes.

Though we will not use object-oriented programming in Python intensively, we still need some basic understanding o
Python classes. Let's just dig into a class-demo, here is `MyClass` in Python

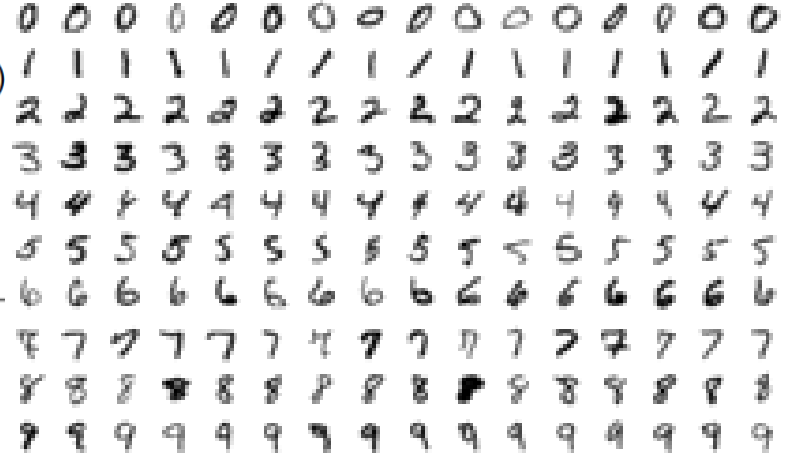
```
class MyClass:
    myvar = "blah"

    def myfun(self):
        print("This is a message inside the class.")

myobjectx = MyClass()
```

Exercise: `L03/datasets.ipynb`

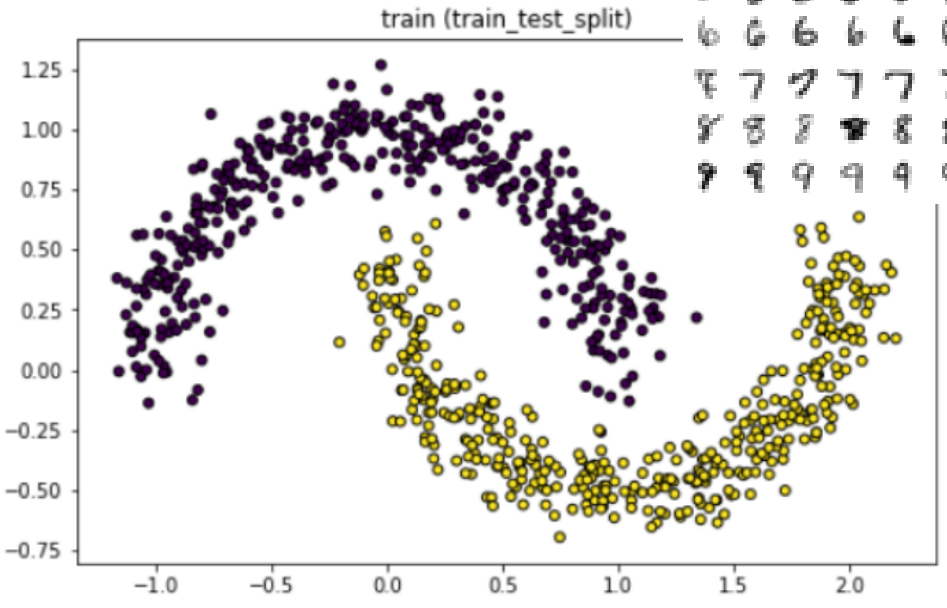
Vanilla Datasets

There are a number of popular datasets out-there, that are used again and again for small scale testing in ML: most popular are Moon, MNIST, Iris and CIFAR(10/100). We will use the three first here.

(More on ML datasets: https://en.wikipedia.org/wiki/List_of_datasets_for_machine_learning_research)



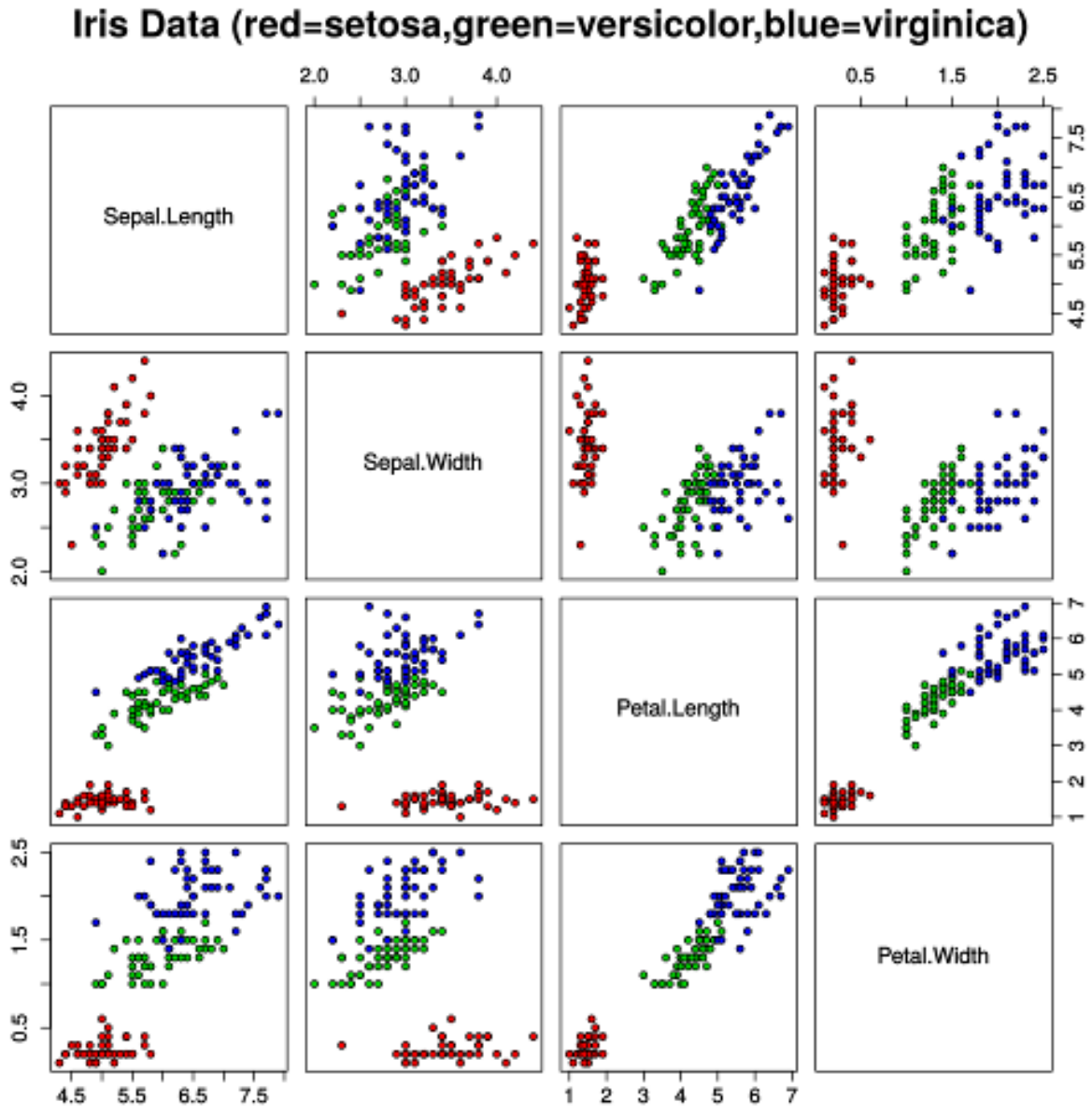
Moon



Vanilla/ML hello-world datasets: Moon, MNIST, iris...

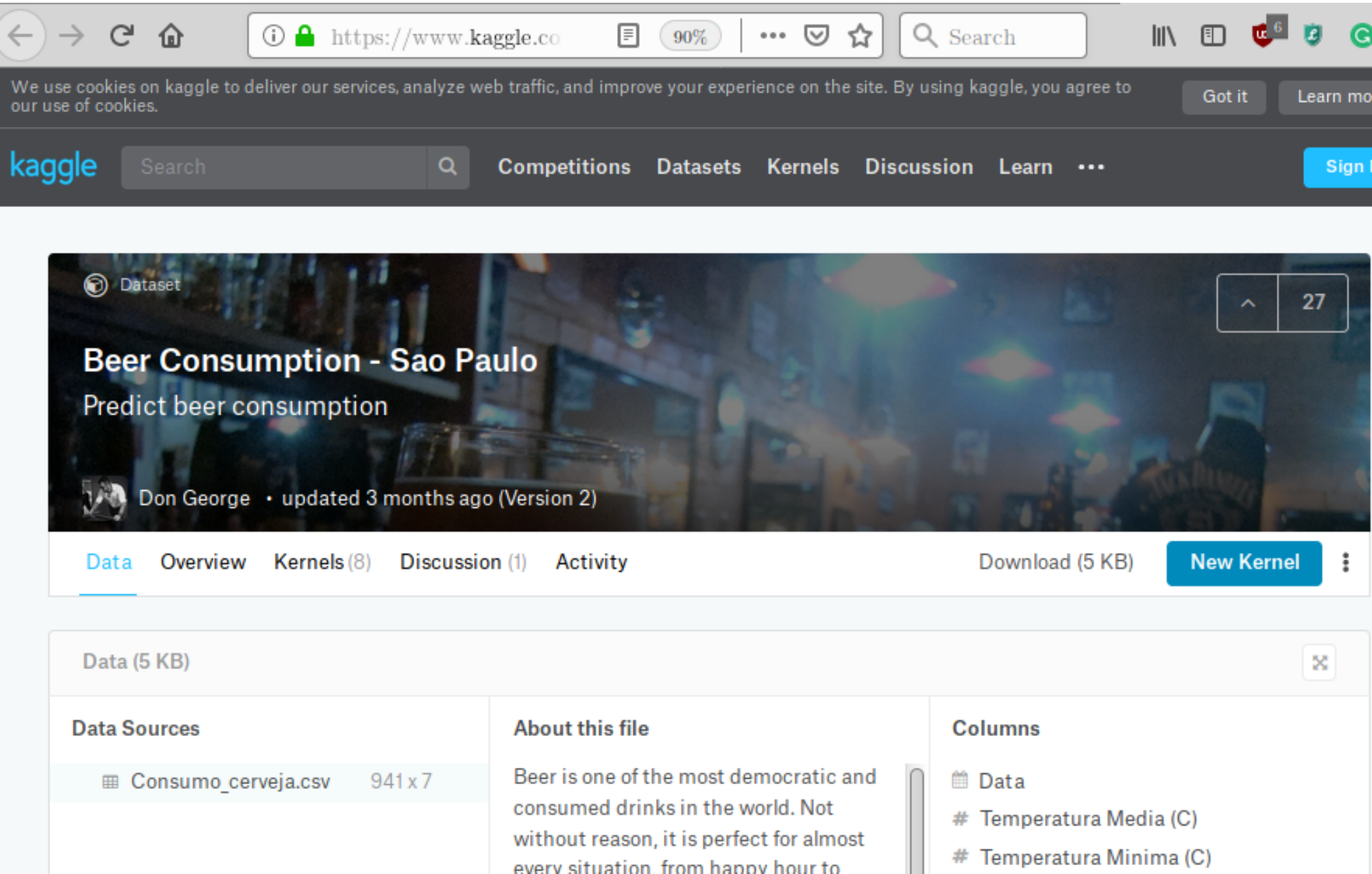
Exercise: `L03/datasets.ipynb`

Feature scatterplot...



Exercise: `L03/datasets.ipynb`

Your dataset, from `https://www.kaggle.com`...



RESUMÉ: Scikit-learn

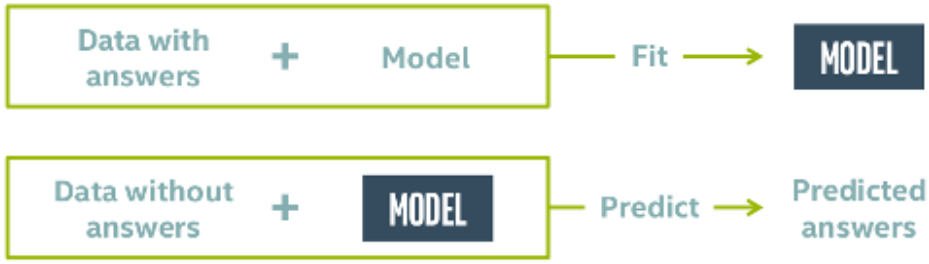
[https://en.wikipedia.org/wiki/Scikit-learn]

Scikit-learn *(formerly scikits.learn) is a free software machine learning library for the Python programming language.*

It features various **classification**, **regression** *and* **clustering** *algorithms [...], and is designed to interoperate with the* **Python** *numerical and scientific libraries NumPy and SciPy.*

The Scikit-learn Fit-Predict Interface

The API has one predominant object: **the estimator.**

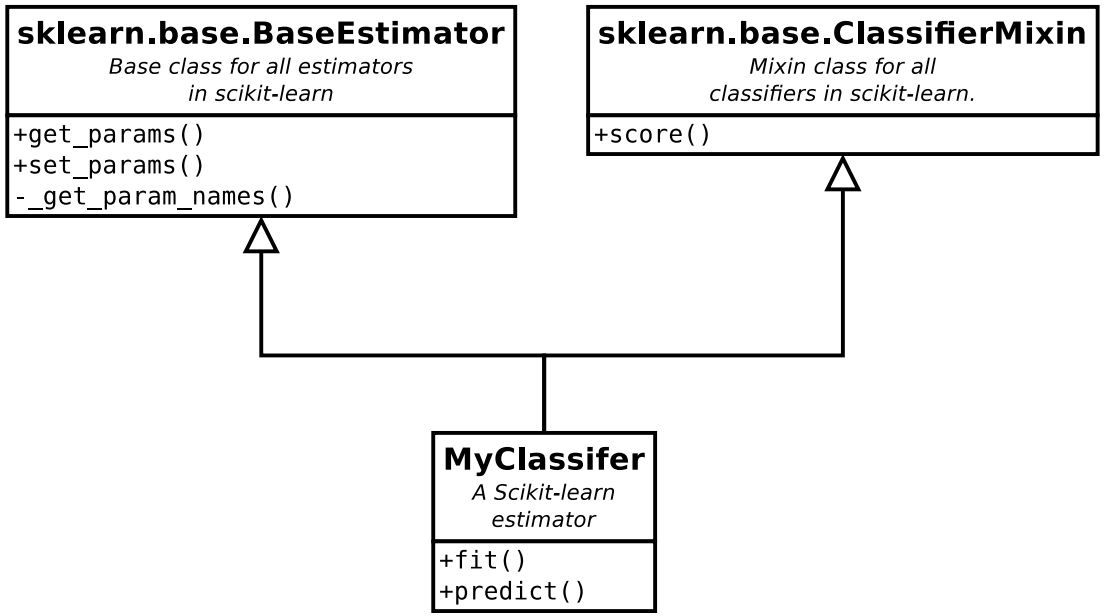


An estimator is an object that fits a model based on some training data and is capable of inferring some properties on new data. It can be, for instance, a classifier or a regressor.

All estimators implement the fit method: `estimator.fit(X,y)` *All built-in estimators also have a* `set_params` *method, which sets data-independent parameters (overriding previous parameter values passed to* `__init__`*.*

All estimators in the main scikit-learn codebase should inherit from `sklearn.base.BaseEstimator`*.*

The Scikit-learn Fit-Predict Interface



Python module and class function and member encapsulation:

- module private: one underscore
- class-private: two underscores

via mangled names.

...NOTE: no `virtual void fit() = 0;` declaration in python!

...for modules, private funs can still be accessed via a hack?!

...src file: `/opt/anaconda3/pkgs/.../sklearn/base.py`

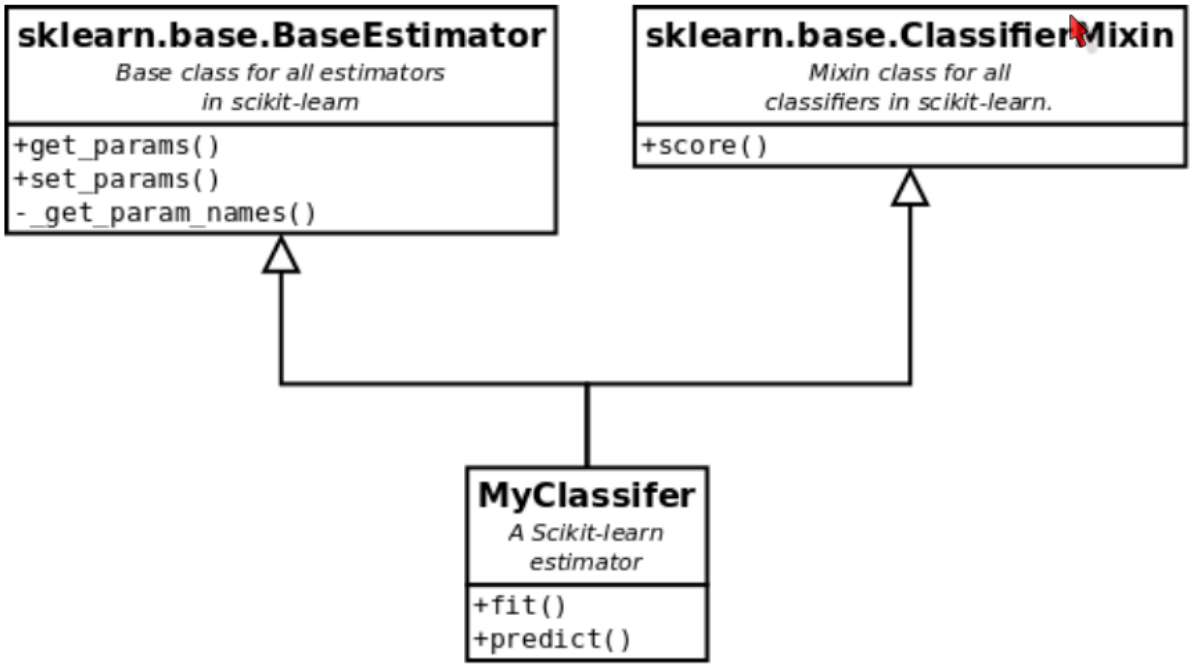
Exercise: `L03/dummy_classifier.ipynb`

A dummy classifier for the fit-predict interface, plus intro to a Stochastic Gradient Decent method (SGD)

Qb Implement a dummy binary classifier

Follow the code found in [HOML], p84, but name you estimator `DummyClassifier` instead of `Never5Classifyer`.

Here our Python class knowledge comes into play. The estimator class hierarchy looks like



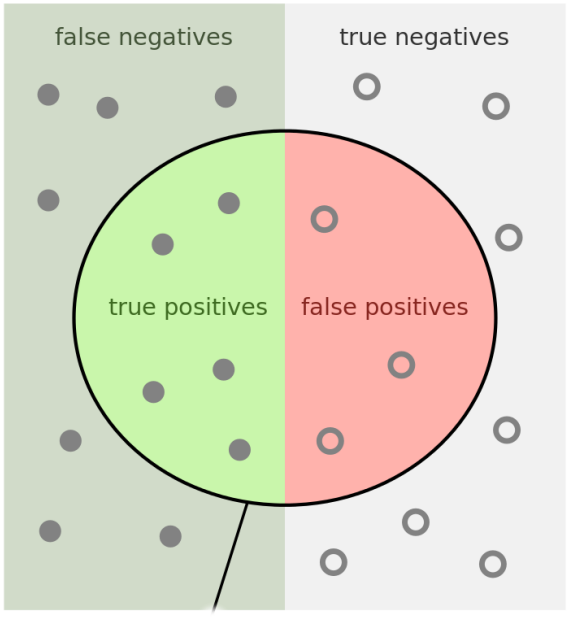
All Scikit-learn classifiers inherit form `BaseEstimator` (and possible also `ClassifierMixin`), and they must have a `fit-predict` function pair (strangely not in the base class!) and you can actually find the `sklearn.base.BaseEstimator` and `sklearn.base.ClassifierMixin` python source code somewhere in you anaconda install dir, if you should have the nerves to go to such interesting details

Exercise: `L03/metrics.ipynb`

Nomenclature

For a binary classifier

NAME	SYMBOL	ALIAS
true positives	*TP*	
true negatives	*TN*	
false positives	*FP*	type I error
false negatives	*FN*	type II error

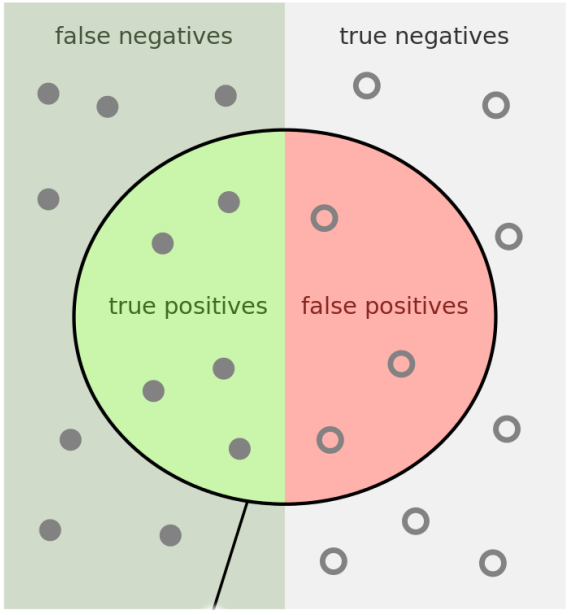


and $N = N_P + N_N$ being the total number of samples and the number of positive and negative samples respectively.

`[https://en.wikipedia.org/wiki/Precision_and_recall]`

Exercise: `L03/metrics.ipynb`

Precision, recall and accuracy, F_1-score, and confusion matrix

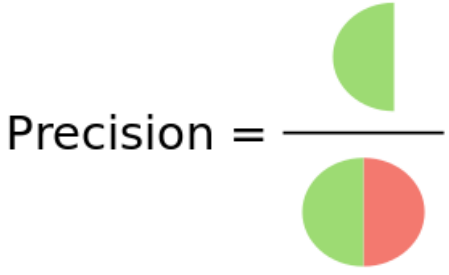


precision, $p = \frac{TP}{TP+FP}$

recall (or sensitivity), $r = \frac{TP}{TP+FN}$

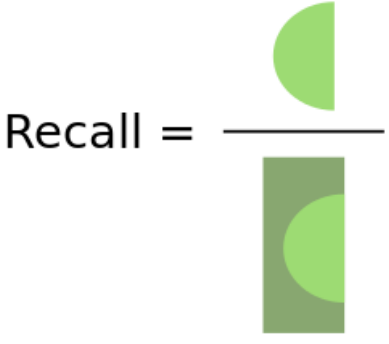
accuracy, $a = \frac{TP+TN}{TP+TN+FP+FN}$

F_1-score, $F_1 = \frac{2pr}{p+r}$



Confusion Matrix, $\mathbf{M}_{\text{confusion}} =$

	actual true	actual false
predicted true	*TP*	*FP*
predicted false	*FN*	*TN*



Exercise: L03/metrics.ipynb

Accuracy Paradox...

```
class ParadoxClassifier(BaseEstimator):
    def fit(self, X, y=None):
        pass
    def predict(self, X):
        return np.ones(len(X),dtype=bool)
```

Test via the breast cancer Wisconsin dataset...

```
X, y_true = load_breast_cancer(return_X_y=True)

print(f"  X.shape={X.shape}, y_true.shape={y_true.shape}")
X_train, X_test, y_train, y_test = train_test_split(X, y_true,
    test_size = 0.2, shuffle = True, random_state= 42)

clf = ParadoxClassifier()
clf.fit(X_train, y_test)
y_pred = clf.predict(X_test)

a = accuracy_score(y_pred, y_test)
print('  acc=', a, ', N=', y_pred.shape[0])
```

prints: `acc= 0.6228070175438597 , N= 114`