



## LESSON 2: Classification

Cost function, Supervised classification, Performance metrics

CARSTEN EIE FRIGAARD

SPRING 2020

$$\mathbf{X} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \dots & x_d^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \dots & x_d^{(2)} \\ \vdots & \vdots & & \vdots \\ x_1^{(n)} & x_2^{(n)} & \dots & x_d^{(n)} \end{bmatrix} = \begin{bmatrix} (\mathbf{x}^{(1)})^T \\ (\mathbf{x}^{(2)})^T \\ \vdots \\ (\mathbf{x}^{(n)})^T \end{bmatrix}$$



# Agenda

Supervised classification, Cost function, Performance metrics

1. Admin (afleveringer, grupper, etc.)
2. Resume
  - ▶ kort om Python libs
  - ▶ supervised learning: en oversigt
3. Lineær algebra og cost funktionen,  $J$ 
  - ▶ matricer, vektors, norms og NumPy,
  - ▶ MSE, MAE,
  - ▶ Jupyter notebook: [L02/cost\\_function.ipynb](#)
4. Supervised binær klassifikation
  - ▶ 'demo' datasæt,
  - ▶ fundamental ML supervised lærings-proces,
  - ▶ Scikit-learn fit-predict interface,
  - ▶ Jupyter notebook: [L02/dummy\\_classifier.ipynb](#)
5. Performance metrics
  - ▶ precision, recall, accuracy,  $F_1$ -score,
  - ▶ confusion matrix,
  - ▶ Jupyter notebook: [L02/performance\\_metrics.ipynb](#)

# The toolset for ML

A list of our toolbox

- ▶ **Python:** our preferred language for ML,
- ▶ **Anaconda:** a particular distribution of python, that we will use,
- ▶ **Jupyter** notebooks: interactive coding and visualization for python (alt: Spider, PyCharm),
- ▶ **NumPy, SciPy, Pandas, Matplotlib, Seaborn:** numerical computation and data visualization libraries for python,
- ▶ **Scikit-learn:** machine learning tools.

# Jupyter Crash Course

Jupyter need-to-know:

- ▶ Ctrl+Enter: executes cell,
- ▶ Shift+Tab: help for function under cursor,
- ▶ Shift+Tab repeated: extended help,
- ▶ Tab: 'tab'-completion??

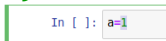
Jupyter magic commands:

- ▶ `%matplotlib inline`: pull in the matplotlib,
- ▶ `%reset -f`: reset all vars (or `-sf`),
- ▶ `%run filename.ipynb`: execute code from another notebook or python file,
- ▶ `%load filename.py`: copy contents of the file and paste into the cell,
- ▶ `! dir`: executes a shell command.

# Jupyter Crash Course

Jupyter shortcuts:

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



```
In [ ]: a=1
```

- ▶ 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

# 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:

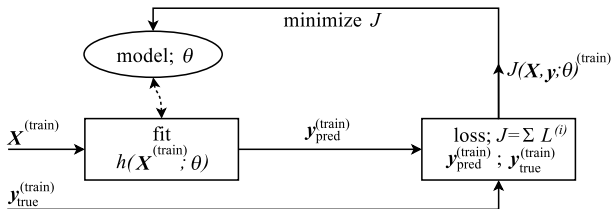
- ▶ `pandas`: python data analysis library, a module for loading/saving and handling large data set,
- ▶ `scipy`: python library used for scientific computing and technical computing.

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



# RESUMÉ

## Data-flow model for supervised learning



$\mathbf{X}^{(\text{train})}$ : trænings data input,

loose notation:  $\mathbf{X}^{(\text{train})} = \mathbf{X}^{(i)}$  for  $\forall i \in \text{train set}$

$\theta$ : model parametre,

$h$ : hypothesis function; types of ML algos,

$\mathbf{y}_{\text{true}}^{(\text{train})}$ : training data output,

$\mathbf{y}_{\text{pred}}^{(\text{train})}$ : predicted (train) data output,

$L^{(i)}$ : individual loss (distance),

$J$ : loss/cost/error/objective function (summeret)



## Exercise: L02/cost\_function.ipynb

### Matrix notation

Say, we have  $d$  features for a given sample point. This  $d$ -sized feature column vector for a data-sample  $i$  is then given by

$$\mathbf{x}^{(i)} = \begin{bmatrix} x_1^{(i)} & x_2^{(i)} & \cdots & x_d^{(i)} \end{bmatrix}^T$$

The full data matrix  $\mathbf{X}$  and target column vector  $\mathbf{y}$  are then constructed out of  $n$  samples of these feature vectors

$$\mathbf{X} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \cdots & x_d^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \cdots & x_d^{(2)} \\ \vdots & & & \vdots \\ x_1^{(n)} & x_2^{(n)} & \cdots & x_d^{(n)} \end{bmatrix} = \begin{bmatrix} (\mathbf{x}^{(1)})^T \\ (\mathbf{x}^{(2)})^T \\ \vdots \\ (\mathbf{x}^{(n)})^T \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix}$$

(and  $\mathbf{X}$  and  $\mathbf{y}$  are sometimes concatenated into a single matrix!)

## Exercise: L02/cost\_function.ipynb

### Distance/norms

The  $\mathcal{L}_2$  Euclidian norm for a vector of size  $n$  is defined as

$$\mathcal{L}_2 : ||\mathbf{x}||_2 = \left( \sum_{i=1}^n |x_i|^2 \right)^{1/2}$$

and thus via linear algebra and vector inner-dot product

$$\mathcal{L}_2^2 : ||\mathbf{x}||_2^2 = \mathbf{x}^\top \mathbf{x}$$

The distance between two vectors is given by

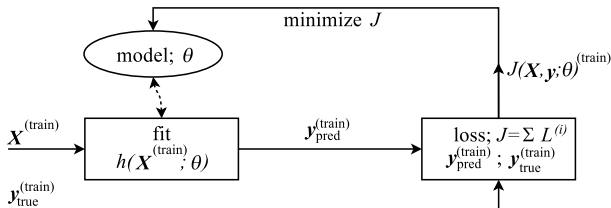
$$\begin{aligned} d(\mathbf{x}, \mathbf{y}) &= ||\mathbf{x} - \mathbf{y}||_2 \\ &= \left( \sum_{i=1}^n |x_i - y_i|^2 \right)^{1/2} \end{aligned}$$

The general  $\mathcal{L}_p$  norm is given by

$$\mathcal{L}_p : ||\mathbf{x}||_p = \left( \sum_i |x_i|^p \right)^{1/p} ; \text{ norm: } \begin{cases} \mathcal{L}_p(\mathbf{x}) = 0, \Rightarrow \mathbf{x} = \mathbf{0} \\ \mathcal{L}_p(\mathbf{x} + \mathbf{y}) \leq \mathcal{L}_p(\mathbf{x}) + \mathcal{L}_p(\mathbf{y}), \\ \quad \quad \quad \text{(triangle inequality)} \\ \mathcal{L}_p(\alpha \mathbf{x}) = |\alpha| \mathcal{L}_p(\mathbf{x}) \end{cases}$$

# Exercise: L02/cost\_function.ipynb

Data-flow model for supervised learning



Express  $J$  in terms of vectors and matrices using the  $\mathcal{L}_2$

$$\begin{aligned} J(\mathbf{X}, \mathbf{y}; \theta) &= \frac{1}{n} \sum_{i=1}^n L^{(i)} \\ &= \frac{1}{n} \sum_{i=1}^n d(h(\mathbf{X}^{(i)}) - \mathbf{y}_{true}^{(i)})^2 \\ &= \frac{1}{n} \|\mathbf{h}(\mathbf{X}) - \mathbf{y}_{true}\|_2^2 \\ &= \frac{1}{n} \|\mathbf{y}_{pred} - \mathbf{y}_{true}\|_2^2 \end{aligned}$$

arriving at a  $J$  proportional to the MSE or  $\mathcal{L}_2$  metric

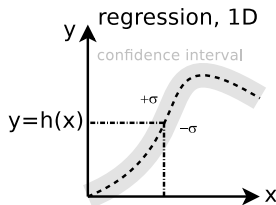
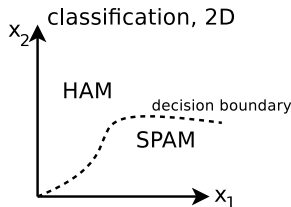
$$\text{cost function: } J(\mathbf{X}, \mathbf{y}_{true}; \theta) \propto \frac{1}{2} \|\mathbf{y}_{pred} - \mathbf{y}_{true}\|_2^2 \propto \text{MSE}$$

# Classification vs. Regression

Given the following hypothesis function

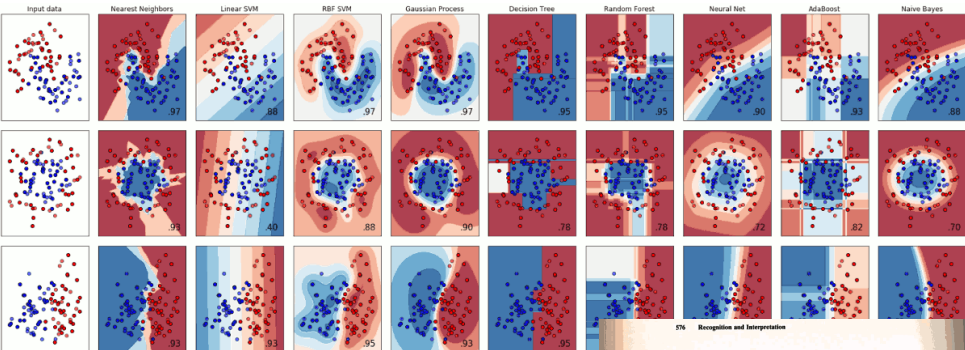
$$h(\mathbf{x}) \rightarrow y$$

- ▶ 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 Boundaries for different Models and Datasets



576 Recognition and Interpretation

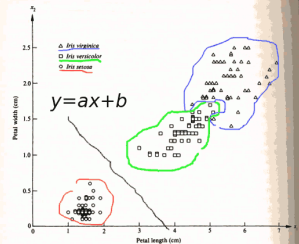


Figure 9.2 Two measurements performed on three types of iris. (Adapted from Duda et al., 1997)

# 'Demo' datasæt

MNIST, Iris og Moon

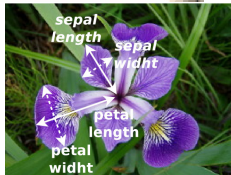
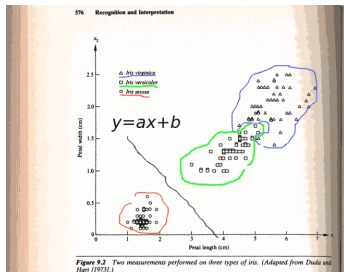
Iris:

Sepal/petal længde/bredde,

Mr. Fisher, 1936,

"Anderson's Iris data set"

`sklearn.datasets.load_iris(..)`



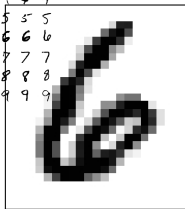
MNIST:

Håndskrevne tal,

preprocesseret, centrerede,

`sklearn.datasets.fetch_openml('mnist_784',..)`

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2  
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3  
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4  
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5  
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6  
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7  
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8  
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9

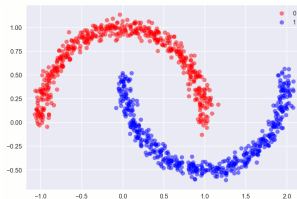


Moon:

'XOR' lign.,

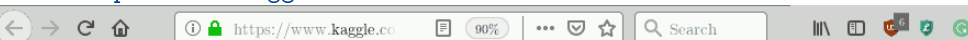
non-linear decision boundary,

`sklearn.datasets.make_moons(..)`



# 'Dit' datasæet

Fra <https://www.kaggle.com...>



We use cookies on kaggle to deliver our services, analyze web traffic, and improve your experience on the site. By using kaggle, you agree to our use of cookies.

Got it

Learn more

kaggle

Search



Competitions

Datasets

Kernels

Discussion

Learn

...

Sign in

Dataset

## Beer Consumption - Sao Paulo

Predict beer consumption



Don George · updated 3 months ago (Version 2)

Data

Overview

Kernels (8)

Discussion (1)

Activity

Download (5 KB)

New Kernel



Data (5 KB)



### Data Sources

Consumo\_cerveja.csv 941 x 7

### About this file

Beer is one of the most democratic and consumed drinks in the world. Not without reason, it is perfect for almost every situation, from happy hour to

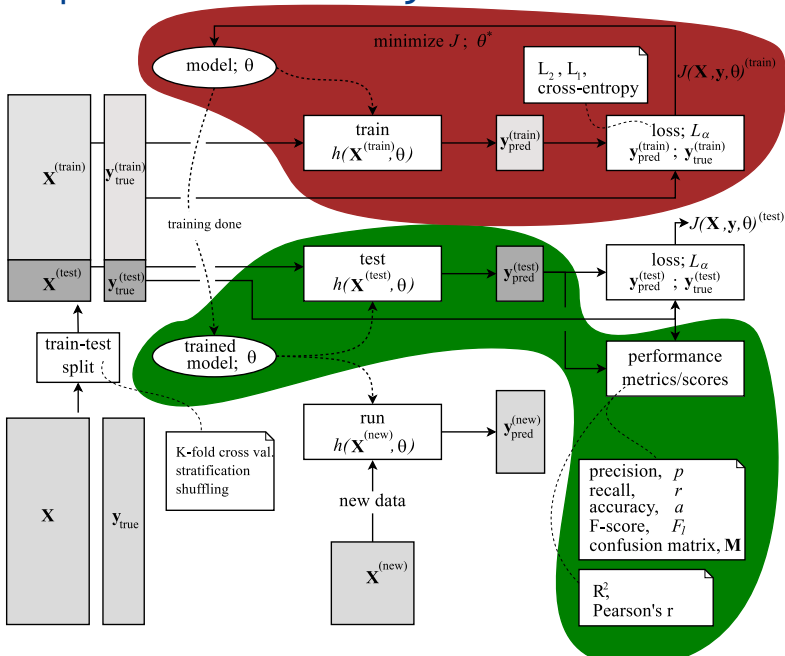
### Columns

Data

# Temperatura Media (C)

# Temperatura Minima (C)

# ML Supervised Learning, Train/Test





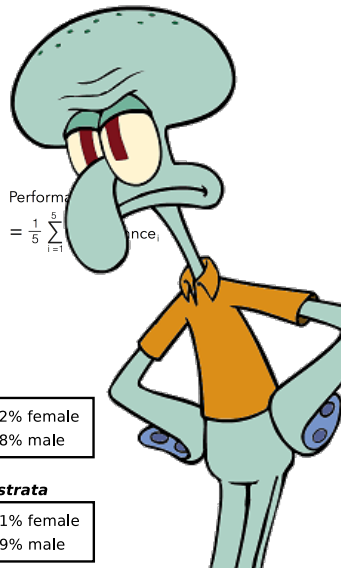
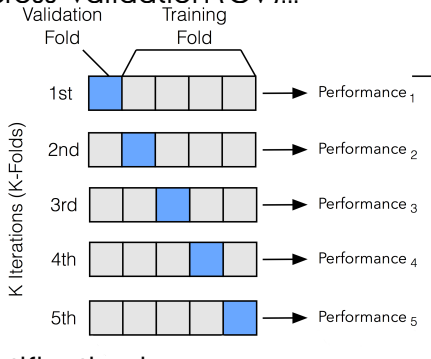
# Fundamental supervised learning-proces

- i) Forbered data:
  - ▶ manuel preprocessing + visualisering (støj, outliers..)
  - ▶ label  $\mathbf{y}_{\text{true}}$  data!!!
  - ▶ normalization, skalering
  - ▶ shuffle,
  - ▶ (stratification, K-fold cross-validation).
- ii) **Split** data i **train/test**.
  - ▶ analogi: skriftlig eksamenssæt på ASE: **test**-træningssæt (eksamen) udleveres ikke til *træning* inden!
- iii) **Træn** på **trænings**-data (**fit**)
  - ▶ ML træning via  $J$ ,
- iv) **Evaluér** på **test**-data (**predict**)
  - ▶ performance metrics/scores

# Forbered data: cross-validation, stratification

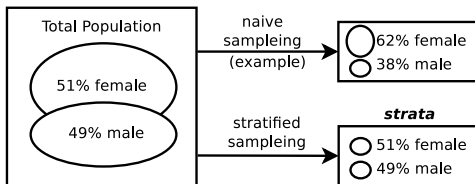
Bemærk: mere preprocess og k-fold cross-validation i L06, koncepter II..

## K-fold cross-validation (CV)...



$$\text{Performance} = \frac{1}{5} \sum_{i=1}^5 \text{Performance}_i$$

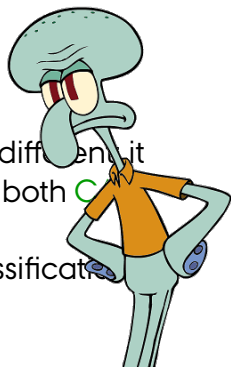
and stratification is...



# Multiclass/Multinomial Classification

## And 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/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/multilabel multiclass classification



# The Scikit-learn Fit-Predict Interface



## Supervised Classification in practice

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

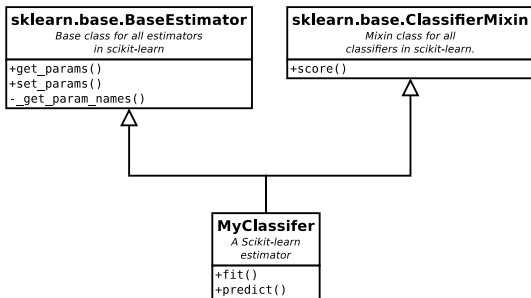


*An estimator is an object that fits a model based on some training data and is capable of inferring some properties on new data. It can be, for instance, a classifier or a regressor.*

*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 funcs can still be accessed via a hack?!

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

# The Scikit-learn Fit-Predict Interface



Demo..

Implementing an estimator via a python class as simple as

```
1 class ParadoxClassifier(BaseEstimator, ClassifierMixin):
2     def fit(self, X, y=None):
3         pass
4     def predict(self, X):
5         assert X.ndim==2
6         return np.ones(X.shape[0],dtype=bool)
```

# Exercise: L02/dummy\_classifier.ipynb

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

The screenshot shows a Jupyter Notebook interface with the title 'dummy\_classifier'. The browser address bar indicates the file is located at 'localhost:8889/notebooks/Downloads/dummy\_classifier.ipynb'. The notebook has a toolbar with various icons for file operations, editing, and running code. Below the toolbar, there is a code cell with the following content:

```
In [ ]: # TODO: add your code here..
        assert False, "TODO: solve Qb, and remove me.."
```

Below the code cell, the text reads: "Qc Implement a dummy binary classifier". This is followed by a paragraph explaining the goal: "Now we will try to create a Scikit-learn compatible estimator implemented via a python class. Follow the code found in [HOML], p84, but name you estimator `DummyClassifier` instead of `Never5Classifier`."

Next, it states: "Here our Python class knowledge comes into play. The estimator class hierarchy looks like". Below this text is a class hierarchy diagram:

```
graph BT
    Base[sklearn.base.BaseEstimator] --> MyClassifier[MyClassifier]
    Mixin[sklearn.base.ClassifierMixin] --> MyClassifier
```

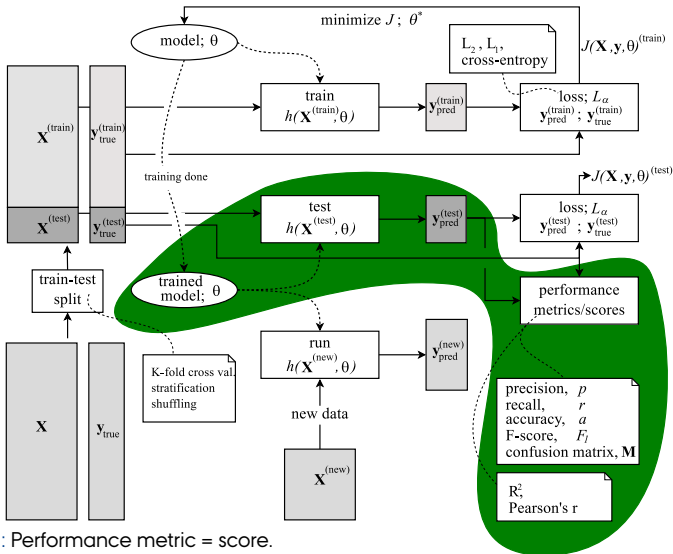
The diagram shows three classes:

- sklearn.base.BaseEstimator**: Base class for all estimators in scikit-learn. Methods: `+get_params()`, `+set_params()`, `- get_param_names()`.
- sklearn.base.ClassifierMixin**: Mixin class for all classifiers in scikit-learn. Method: `+score()`.
- MyClassifier**: A Scikit-learn estimator. Methods: `+fit()`, `+predict()`.

Arrows indicate that `MyClassifier` inherits from both `BaseEstimator` and `ClassifierMixin`.

# Evaluér på test-data: Performance metrics

Kort intro til konceptet *performance metrics*.



NOTE<sub>0</sub>: Performance metric = score.

NOTE<sub>1</sub>: 'Performance measure' begreb bruges ikke, kun score eller perf. metric.

NOTE<sub>2</sub>: Loss er ML algo'ens 'performance mål', score er vores evalueringsmål.



# Exercise: L02/performance\_metrics.ipynb

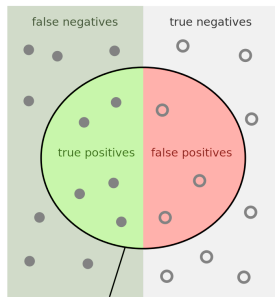
## Nomenclature

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](https://en.wikipedia.org/wiki/Precision_and_recall)]



# Exercise: L02/performance\_metrics.ipynb

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

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

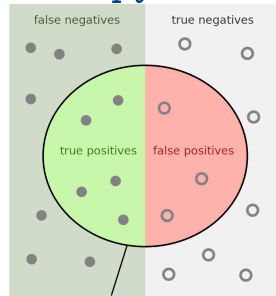
recall (or sensitivity),  $r = \frac{TP}{TP+FN}$

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



Precision =  $\frac{\text{green semi-circle}}{\text{green semi-circle} + \text{red semi-circle}}$

Recall =  $\frac{\text{green semi-circle}}{\text{green semi-circle} + \text{green rectangle}}$


NOTE<sub>0</sub>: you can compare precision... $F_1$ -score, but not necessarily the cost,  $J$ .

NOTE<sub>1</sub>: beware of matrix transpose and interpretation of 'TP/TN'!

# Exercise: L02/performance\_metrics.ipynb

## Nomenclature for the Confusion Matrix

		True condition			
		Condition positive	Condition negative	Prevalence $= \frac{\Sigma \text{Condition positive}}{\Sigma \text{Total population}}$	Accuracy (ACC) = $\frac{\Sigma \text{True positive} + \Sigma \text{True negative}}{\Sigma \text{Total population}}$
Predicted condition	Predicted condition positive	True positive, Power	False positive, Type I error	Positive predictive value (PPV), Precision = $\frac{\Sigma \text{True positive}}{\Sigma \text{Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\Sigma \text{False positive}}{\Sigma \text{Predicted condition positive}}$
	Predicted condition negative	False negative, Type II error	True negative	False omission rate (FOR) $= \frac{\Sigma \text{False negative}}{\Sigma \text{Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\Sigma \text{True negative}}{\Sigma \text{Predicted condition negative}}$
		True positive rate (TPR), Recall, Sensitivity, probability of detection $= \frac{\Sigma \text{True positive}}{\Sigma \text{Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm $= \frac{\Sigma \text{False positive}}{\Sigma \text{Condition negative}}$	Positive likelihood ratio (LR+) $= \frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) $= \frac{\text{LR+}}{\text{LR-}}$  F1 score = $\frac{2}{\frac{1}{\text{Recall}} + \frac{1}{\text{Precision}}}$
		False negative rate (FNR), Miss rate $= \frac{\Sigma \text{False negative}}{\Sigma \text{Condition positive}}$	True negative rate (TNR), Specificity (SPC) $= \frac{\Sigma \text{True negative}}{\Sigma \text{Condition negative}}$	Negative likelihood ratio (LR-) $= \frac{\text{FNR}}{\text{TNR}}$	

Mr. Itmal  : prevalence, positive predictive value, etc.  
not important to know at all!

# Exercise: L02/performance\_metrics.ipynb

## Accuracy Paradox...

```
1 class ParadoxClassifier(BaseEstimator, ClassifierMixin):
2     def fit(self, X, y=None):
3         pass
4     def predict(self, X):
5         assert X.ndim==2
6         return np.ones(X.shape[0],dtype=bool)
```

## Test via the breast cancer Wisconsin dataset..

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

```
4
5 clf = ParadoxClassifier()
6 clf.fit(X_train, y_train)
7 y_pred = clf.predict(X_test)
```

**prints:** acc=0.6228070175438597,  
N=114

```
8
9 acc = accuracy_score(y_test, y_pred)
10 print(f' acc={acc}, N={y_pred.shape[0]}')
11 score = clf.score(X_test, y_test)
12 print(f' clf.score()={score} (same as accuracy_score)')
```

NOTE<sub>0</sub>: for MNIST, a dum classify as '5'  $\sim a = 10\%$

NOTE<sub>1</sub>: for MNIST, a dum classify not-as '5'  $\sim a = 90\%$

# Exercise: L02/performance\_metrics.ipynb

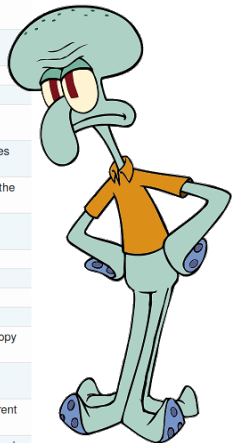
More on metrics, oh-so-many!

[<https://scikit-learn.org/stable/modules/classes.html#sklearn-metrics-metrics>]

## Classification metrics

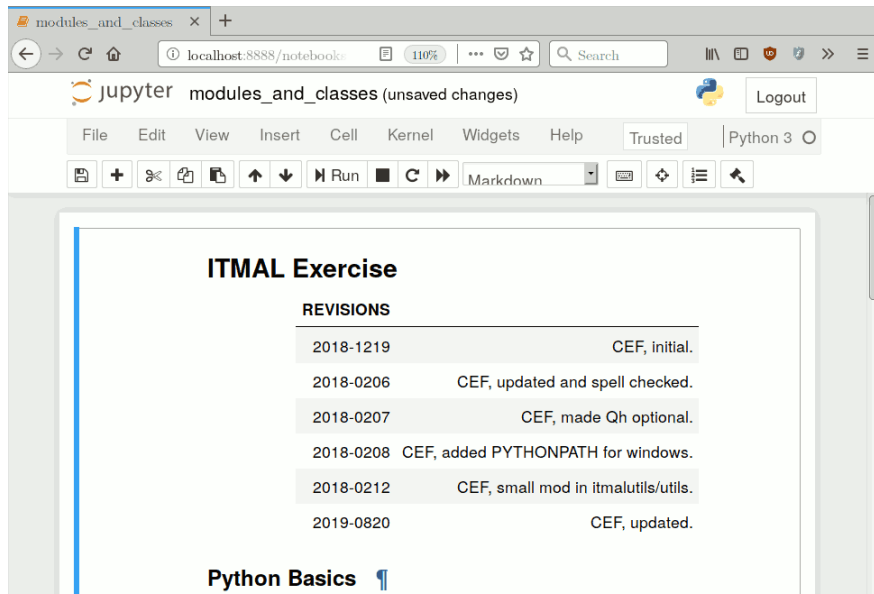
See the [Classification metrics](#) section of the user guide for further details.

<code>metrics.accuracy_score(y_true, y_pred[, ...])</code>	Accuracy classification score.
<code>metrics.auc(x, y[, reorder])</code>	Compute Area Under the Curve (AUC) using the trapezoidal rule
<code>metrics.average_precision_score(y_true, y_score)</code>	Compute average precision (AP) from prediction scores
<code>metrics.balanced_accuracy_score(y_true, y_pred)</code>	Compute the balanced accuracy
<code>metrics.brier_score_loss(y_true, y_prob[, ...])</code>	Compute the Brier score.
<code>metrics.classification_report(y_true, y_pred)</code>	Build a text report showing the main classification metrics
<code>metrics.cohen_kappa_score(y1, y2[, labels, ...])</code>	Cohen's kappa: a statistic that measures Inter-annotator agreement.
<code>metrics.confusion_matrix(y_true, y_pred[, ...])</code>	Compute confusion matrix to evaluate the accuracy of a classification
<code>metrics.f1_score(y_true, y_pred[, labels, ...])</code>	Compute the F1 score, also known as balanced F-score or F-measure
<code>metrics.fbeta_score(y_true, y_pred, beta[, ...])</code>	Compute the F-beta score
<code>metrics.hamming_loss(y_true, y_pred[, ...])</code>	Compute the average Hamming loss.
<code>metrics.hinge_loss(y_true, pred_decision[, ...])</code>	Average hinge loss (non-regularized)
<code>metrics.jaccard_similarity_score(y_true, y_pred)</code>	Jaccard similarity coefficient score
<code>metrics.log_loss(y_true, y_pred[, eps, ...])</code>	Log loss, aka logistic loss or cross-entropy loss.
<code>metrics.matthews_corrcoef(y_true, y_pred[, ...])</code>	Compute the Matthews correlation coefficient (MCC)
<code>metrics.precision_recall_curve(y_true, ...)</code>	Compute precision-recall pairs for different probability thresholds
<code>metrics.precision_recall_fscore_support(...)</code>	Compute precision, recall, F-measure and support for each class



# Exercise: L02/modules\_and\_classes.ipynb

## Modules and Packages...

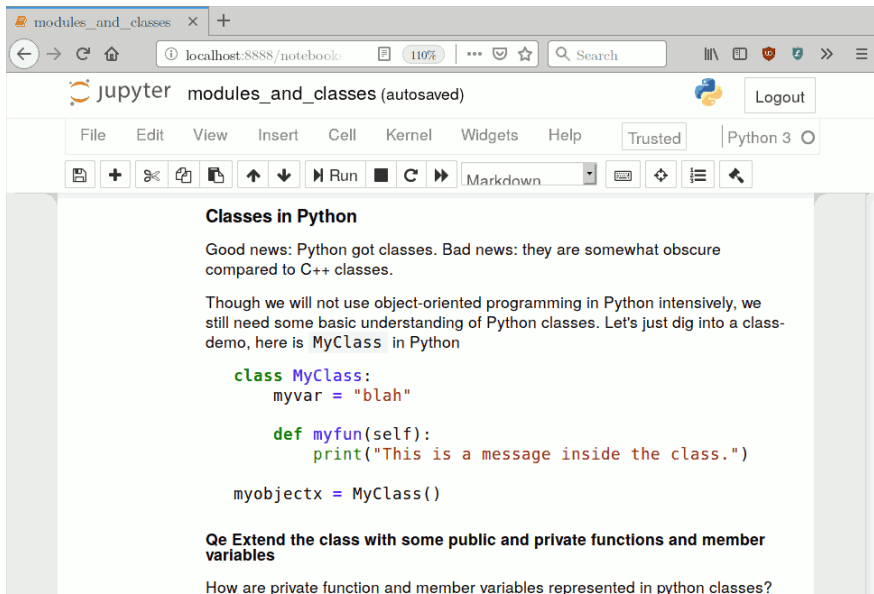


The screenshot shows a Jupyter Notebook interface with the following elements:

- Browser Tab:** modules\_and\_classes
- Address Bar:** localhost:8888/notebooks
- Jupyter Header:** modules\_and\_classes (unsaved changes) with a Python logo and a Logout button.
- Menu Bar:** File, Edit, View, Insert, Cell, Kernel, Widgets, Help. Status: Trusted, Python 3.
- Toolbar:** Includes icons for saving, adding cells, undo, redo, and running code.
- Notebook Content:**
  - ITMAL Exercise**
  - REVISIONS**
  - |           |                                     |
|-----------|-------------------------------------|
| 2018-1219 | CEF, initial.                       |
| 2018-0206 | CEF, updated and spell checked.     |
| 2018-0207 | CEF, made Qh optional.              |
| 2018-0208 | CEF, added PYTHONPATH for windows.  |
| 2018-0212 | CEF, small mod in itmalutils/utils. |
| 2019-0820 | CEF, updated.                       |
  - Python Basics**

# Exercise: L02/modules\_and\_classes.ipynb

Python classes...



The screenshot shows a Jupyter Notebook window titled "modules\_and\_classes (autosaved)". The browser address bar shows "localhost:8888/notebooks". The Jupyter interface includes a top bar with the Jupyter logo, the notebook title, a "Logout" button, and a menu bar with options: File, Edit, View, Insert, Cell, Kernel, Widgets, Help. Below the menu bar is a toolbar with icons for saving, adding cells, undo, redo, running, and other actions. The notebook content area displays the following text:

## 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 of 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()
```

### Qe Extend the class with some public and private functions and member variables

How are private function and member variables represented in python classes?

# Exercise: L02/datasets.ipynb

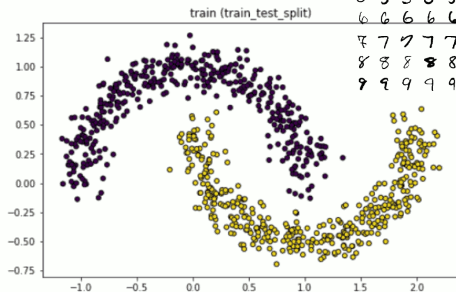
## Vanilla Datasets

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](https://en.wikipedia.org/wiki/List_of_datasets_for_machine_learning_research))

## Moon

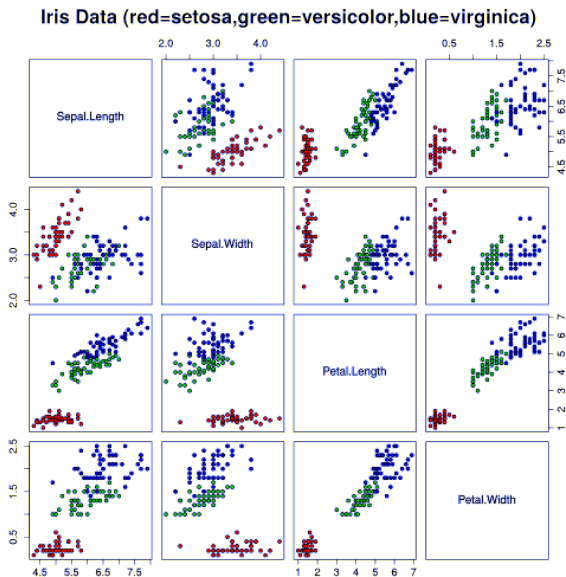


Vanilla/ML hello-world datasets: Moon, MNIST, iris...



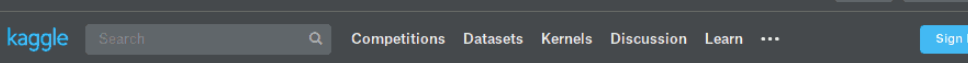
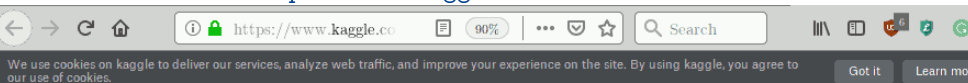
# Exercise: L02/datasets.ipynb


Feature scatterplot...




# Exercise: L02/datasets.ipynb


Your dataset, from <https://www.kaggle.com...>



 Dataset



**Beer Consumption - Sao Paulo**  
Predict beer consumption


 Don George · updated 3 months ago (Version 2)

[Data](#) [Overview](#) [Kernels \(8\)](#) [Discussion \(1\)](#) [Activity](#)

Download (5 KB) [New Kernel](#)

Data (5 KB)

## Data Sources

 Consumo\_cerveja.csv 941 x 7

## About this file

Beer is one of the most democratic and consumed drinks in the world. Not without reason, it is perfect for almost every situation from happy hour to

## Columns

 Data  
# Temperatura Media (C)  
# Temperatura Minima (C)