



## LESSON 2: Classification

Cost function, Supervised classification, Performance metrics

CARSTEN EIE FRIGAARD

AUTUMN 2021

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



# Agenda

## Cost Function, Supervised Classification, Performance Metrics

1. Admin (afleversformat, grupper, etc.)
2. Forelæsning
  - ▶ Resumé
  - ▶ Lineær algebra og cost funktionen,  $J$ 
    - ▶ Opgave: [L02/cost\\_function.ipynb](#)
  - ▶ Fundamental ML supervised lærings-proces,
  - ▶ Supervised binær klassifikation
    - ▶ Opgave: [L02/dummy\\_classifier.ipynb](#)
    - ▶ Scikit-learn fit-predict interface,
  - ▶ Scores/Performance metrics
    - ▶ Opgave: [L02/performance\\_metrics.ipynb](#)
3. Opgaveregning på klassen..

# RESUMÉ: 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.

# RESUMÉ: 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??

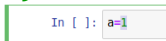
Jupyter magic commands:

- ▶ `%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.

# 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 deprecated!*

# 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>



# THE COST FUNCTION (LOSS)

---

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

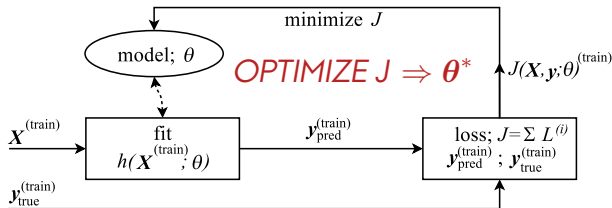
$$\mathcal{L}_2^2 : \quad ||\mathbf{x}||_2^2 = \mathbf{x}^\top \mathbf{x}$$

$$d(\mathbf{x}, \mathbf{y}) = ||\mathbf{x} - \mathbf{y}||_2$$



# The Cost Function

Data-flow model for supervised learning



$X^{(train)}$ : trænings data input,

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

$\theta$ : model parametre,

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

$y_{true}^{(train)}$ : training data output,

$y_{pred}^{(train)}$ : predicted (train) data output,

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

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

## Exercise: L02/cost\_function.ipynb

### The Design Matrix

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

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

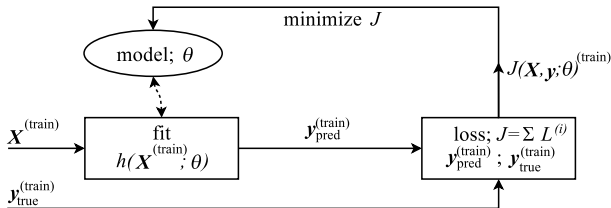
The general  $\mathcal{L}_p$  norm is given by

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

• More generally, the  $\ell_k$  norm of a vector  $\mathbf{v}$  containing  $n$  elements is defined as  $||\mathbf{v}||_k = \left( |v_1|^k + |v_2|^k + \dots + |v_n|^k \right)^{1/k}$ .  $\ell_0$  just gives the number of non-zero elements in the vector, and  $\ell_\infty$  gives the maximum absolute value in the vector.

# Exercise: L02/cost\_function.ipynb

Data-flow model for supervised learning



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

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

Equation 2-1. Root Mean Square Error (RMSE)

$$RMSE(\mathbf{X}, h) = \sqrt{\frac{1}{m} \sum_{i=1}^m (h(\mathbf{x}^{(i)}) - y^{(i)})^2}$$

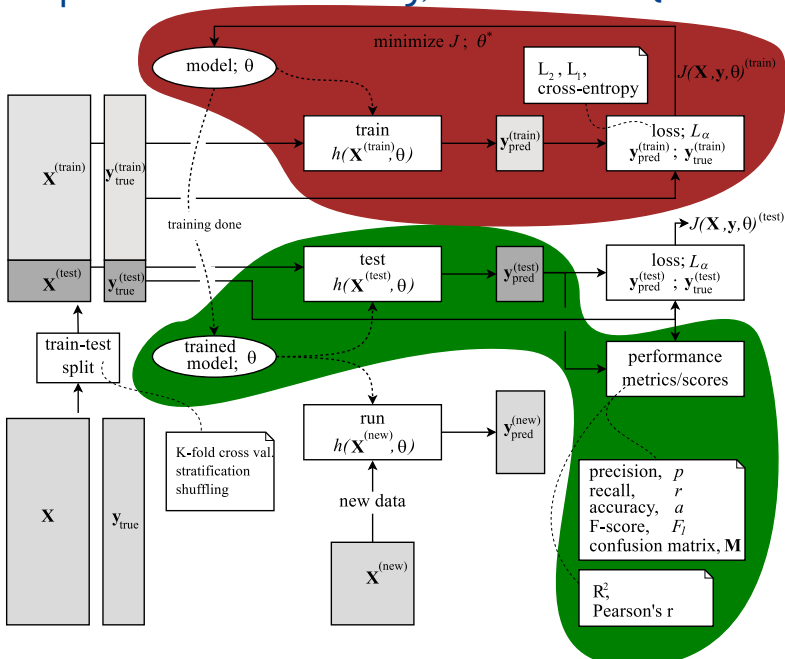
arriving at a  $J$  proportional to the MSE or  $\mathcal{L}_2$  metric

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

# 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

# ML Supervised Learning, Train/Test (The Map)



# CLASSIFICATION

---

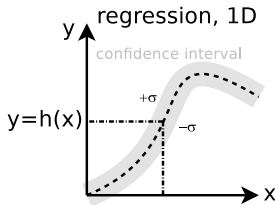
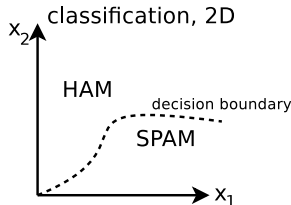
5 0 4 1 9 2 1 3 1  
3 5 3 6 1 7 2 8 6  
4 0 9 1 1 2 4 3 2  
3 8 6 9 0 5 6 0 7  
1 8 1 9 3 9 8 5 9  
3 0 7 4 9 8 0 9 4  
4 4 6 0 4 5 6 1 0

# Classification vs. Regression

Given the following hypothesis function

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

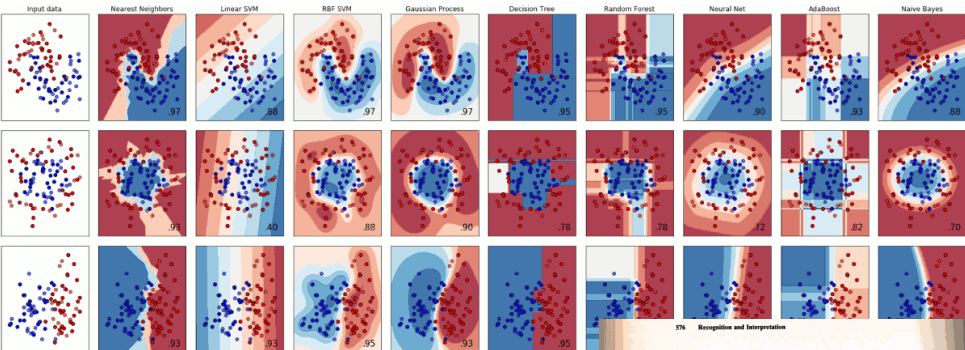
- ▶ if  $y$  is *discrete/categorical* variable, then this is **classification** problem.
- ▶ if  $y$  is *real number/continuous*, then this is a **regression** problem.





# Classification

## Decision Boundaries for different Models and Datasets



576 Recognition and Interpretation

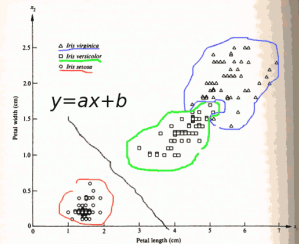
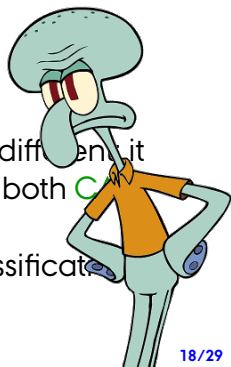


Figure 9.2 Two measurements performed on three types of iris. (Adapted from Duda et al., 1997)

# 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**.*

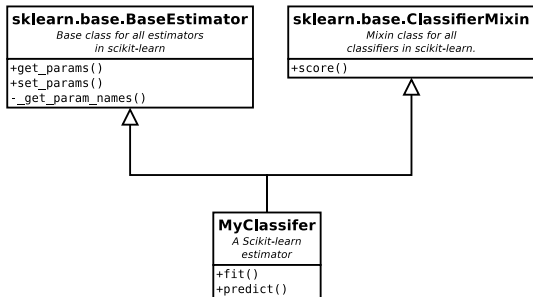


*An estimator is an object that fits a model based on some training data and is capable of inferring some properties on new data. It can be, for instance, a classifier or a regressor.*

*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/pkgsrc/.../sklearn/base.py`

# The Scikit-learn Fit-Predict Interface



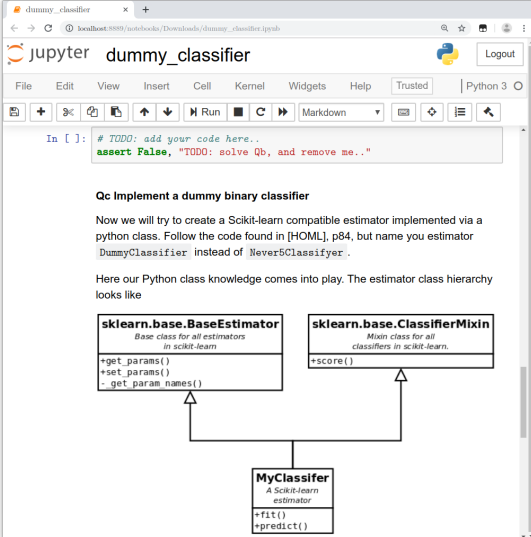
Demo..

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 code cell contains the following text:

```
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**

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 your estimator `DummyClassifier` instead of `Never5Classifier`.

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

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

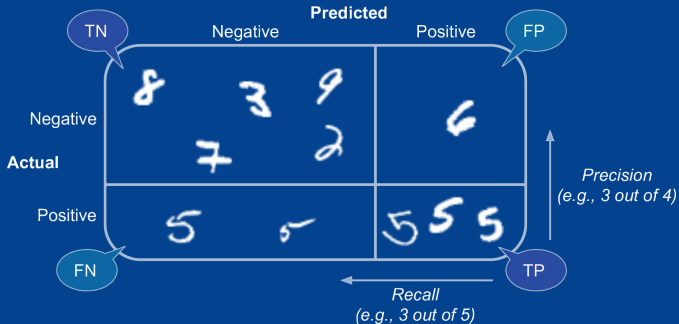
The diagram illustrates the class hierarchy for a Scikit-learn estimator. It shows three classes:

- 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 inheritance: `MyClassifier` inherits from both `BaseEstimator` and `ClassifierMixin`.

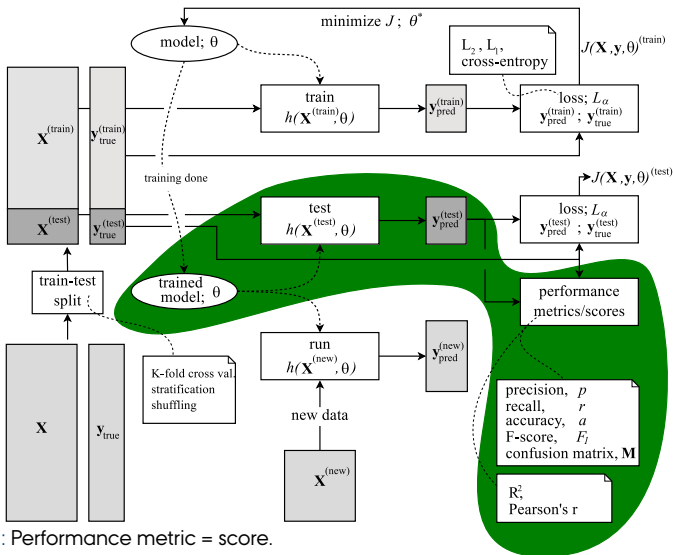
# PERFORMANCE METRICS (SCORES)

---



# 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.



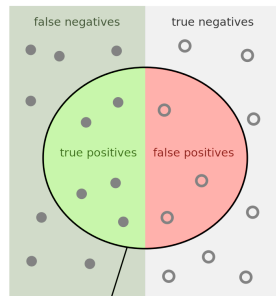
# Exercise: L02/performance\_metrics.ipynb

## 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

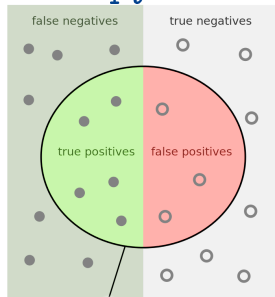
Precision, recall and accuracy,  $F_1$ -score, and confusion matrix

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,  $M_{\text{confusion}} =$

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

Precision = 



Recall = 



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!

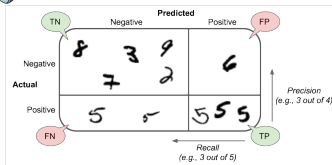


Figure 3-2. An illustrated confusion matrix

# 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 X_train, X_test, y_train, y_test =
2     train_test_split(
3         X, y_true, test_size=0.2, shuffle=True, random_state=42
4     )
5
6 clf = ParadoxClassifier()
7 clf.fit(X_train, y_train)
8 y_pred = clf.predict(X_test)
9
10 acc = accuracy_score(y_test, y_pred)
11 print(f' acc={acc}, N={y_pred.shape[0]}')
12 score = clf.score(X_test, y_test)
13 print(f' clf.score()={score} (same as accuracy_score)')
```

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

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

