AARHUS
UNIVERSITY
SCHOOL OF ENGINEERING

LESSON 2: Concepts I

Cost function, Supervised classification, Performance metrics

CARSTEN EIE FRIGAARD

AUTUMN 2019

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

Agenda

Koncepter I: Supervised classification, Cost function, Performance metrics

1. Opgave afleverings-note + ITMAL grupper..
2. Python/Anaconda/Jupyter noter+crash course.
3. Linær algebra og cost funktionen, J
   - matricer, vektors, norms og `NumPy`,
   - MSE, MAE,
   - Jupyter notebook: `L02/cost_function.ipynb`
4. Klassifikation
   - 'demo' datasæt,
   - fundamental ML supervised lærings-proces,
   - binær klassifikation,
   - Scikit-learn fit-predict interface,
   - Jupyter notebook: `L02/dummy_classifier.ipynb`
5. Performace 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 prefeered language for ML,
- **Anaconda**: a particular distibution 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 Couse

Jupyter need-to-know:

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

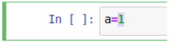
Jupyter magic commands:

- `%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),



- 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 depricated!*

Matplotlib Crash Course

Visualizations can be created in multiple ways:

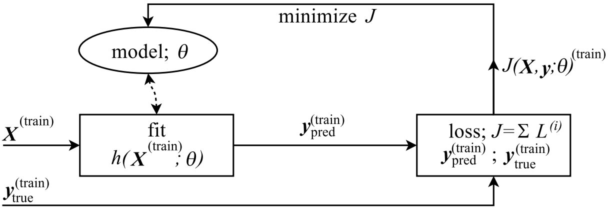
- `matplotlib`
- `pandas`: (via matplotlib),
- `seaborn`: statistically-focused plotting methods.

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

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

RESUMÉ

Data-flow model for supervised learning



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

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

$\boldsymbol{\theta}$: model parametre,

h: hypothesis function; types of ML algos,

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

$\mathbf{y}^{(\text{train})}_{\text{pred}}$: 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 concantenated 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

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

and thus via linear algebra and vector inner-dot product

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

The distance between two vectors is given by

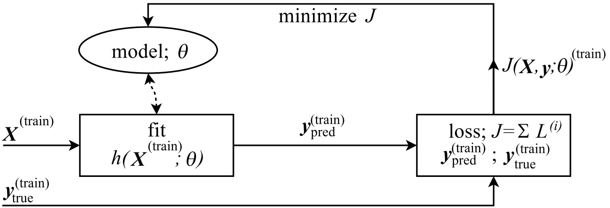
$$\begin{aligned} d(\mathbf{x}, \mathbf{y}) &= ||\mathbf{x} - \mathbf{y}||_2 \\ &= \left(\sum_{i=1}^{n} |x_i - y_i|^2 \right)^{1/2} \end{aligned}$$

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

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

Exercise: `L02/cost_function.ipynb`

Data-flow model for supervised learning



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

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

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

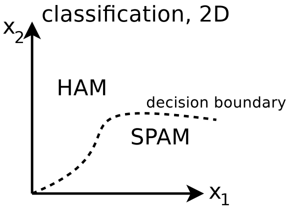
cost function: $J(\mathbf{X},\mathbf{y}_{true};\boldsymbol{\theta}) \propto \frac{1}{2}||\mathbf{y}_{pred} - \mathbf{y}_{true}||_2^2 \propto MSE$

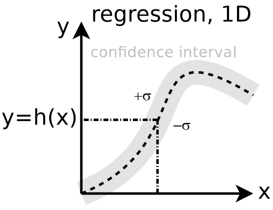
Classification vs. Regression

Given the following

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

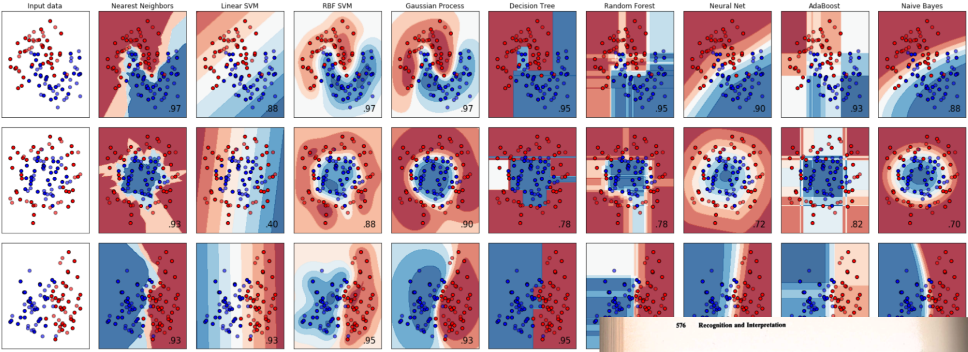
- if y is discrete/categorical variable, then this is **classification** probl
- if y is real number/continuous, then this is a **regression** problem.





Classification

Decision Boundaries for different Models and Datasets



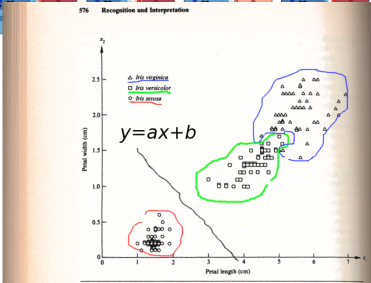


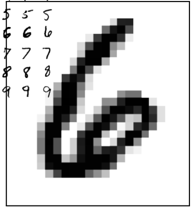
Figure 9.2 Two measurements performed on three types of iris. (Adapted from Duda and Hart [1973].)

'Demo' datasæt

MNIST, Iris og Moon

MNIST:

Håndskrevne tal,
preprocesseret, centrerede,
`sklearn.datasets.fetch_openml('mnist_784'..`



Iris:

Sepal/petal længde/bredde,
Mr. Fisher 1936,
"Anderson's Iris data set"
`sklearn.datasets.load_iris(..)`

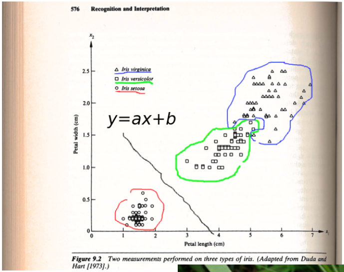
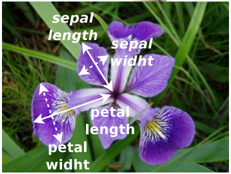
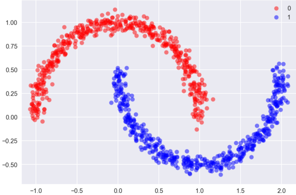


Figure 9.2 Two measurements performed on three types of iris. (Adapted from Duda and Hart [1973].)



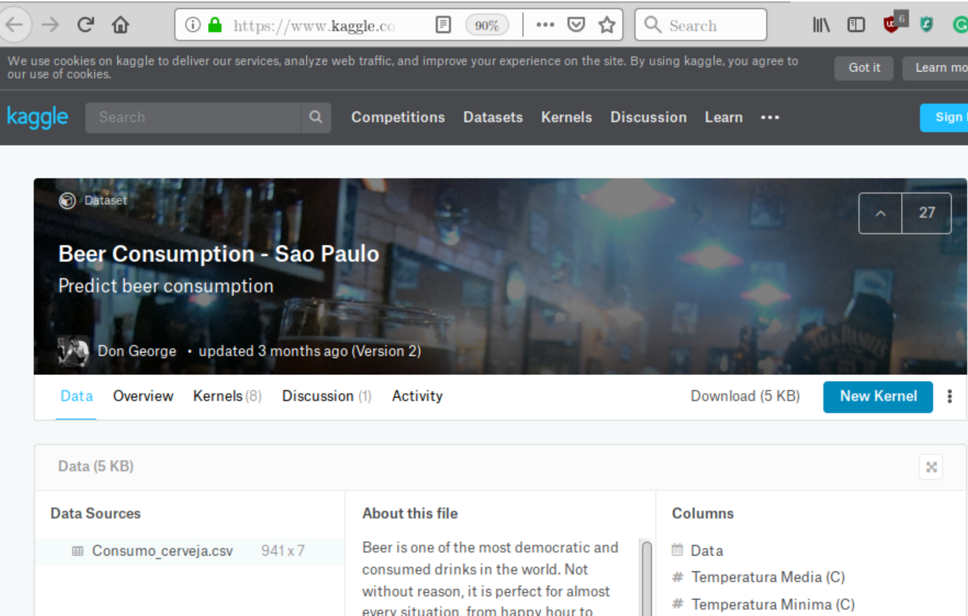
Moon:

'XOR' lign.,
non-linear decision boundary,
`sklearn.datasets.make_moons(..)`

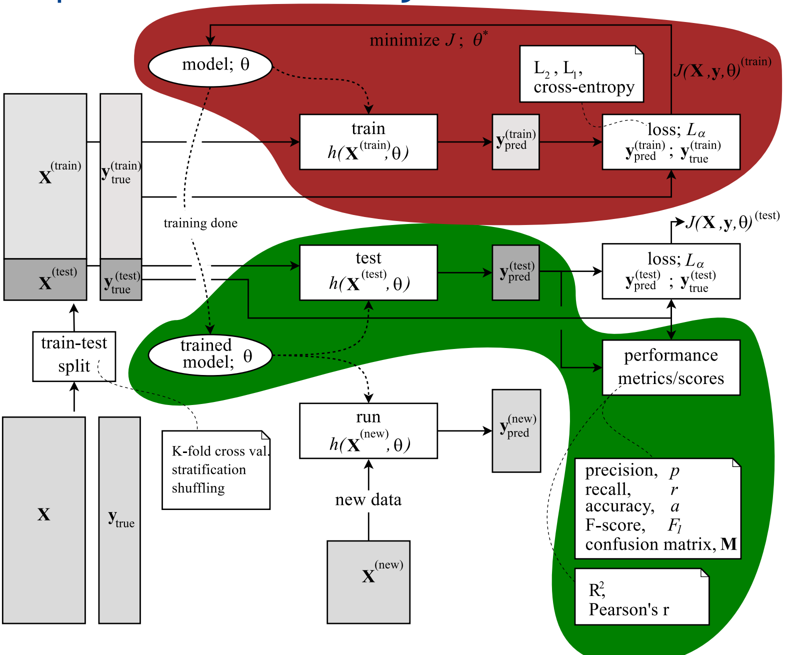


'Dit' datasæt

Fra `https://www.kaggle.com...`



ML Supervised Learning, Train/Test



Fundamental supervised learning-proces

i) Forbered data:
   - manuel preprocessering + visualisering (støj, outliers..)
   - 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!

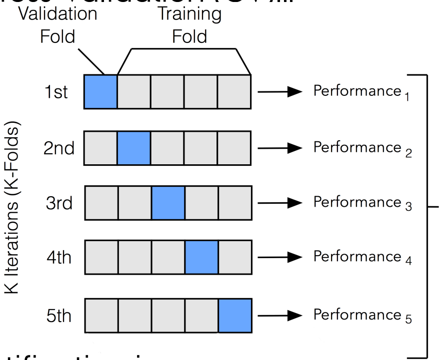
iii) **Træn** på trænings-data (`fit`)
   - ML træning via J,

iv) **Evaluér** på test-data (`predict`)
   - performance metrics/scores

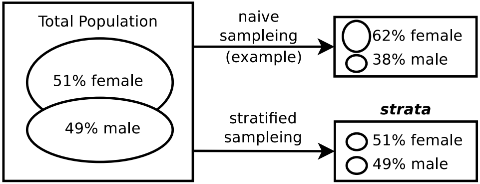
Forbered data: cross-validation, stratification

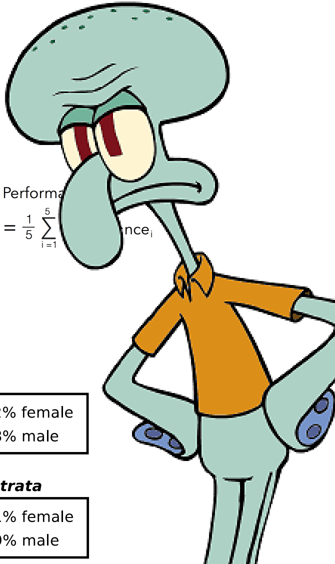
Bemærk: mere preprocess og k-fold cross-validation i L06, koncepter II..

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



and stratification is...

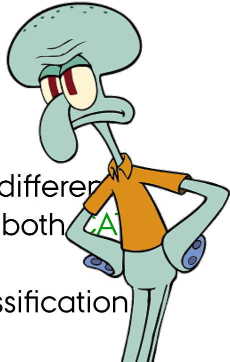




Multiclass/Multinomial Classification

And Introduction to Multilabel Classification

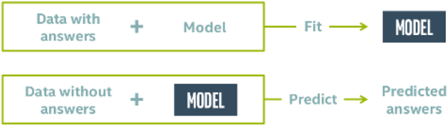
- Many classifiers are binary (HAM/SPAM)
- What to do for say a three category, like
  CAT/DOG/TURTLE
  problem?
- Divide into three CAT/NON-CAT, etc, binary classifiers and solve!
- Aka.: one-vs-rest (OvA),
  one-vs-all (OvA),
  one-against-all (OAA).
- Or the one-vs-one (OvO) method.
- NOTE: Multilabel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

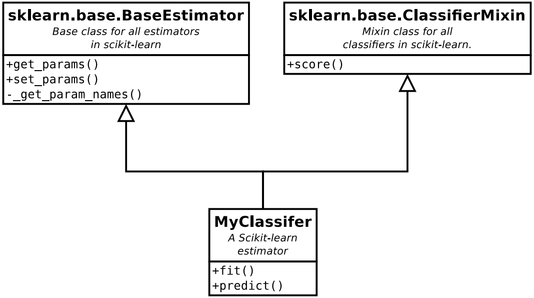


An estimator is an object that fits a model based on some training data and is capable of inferring some properties on new data. It can be, for instance, a classifier or a regressor.

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 funs can still be accessed via a hack?!

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

The Scikit-learn Fit-Predict Interface

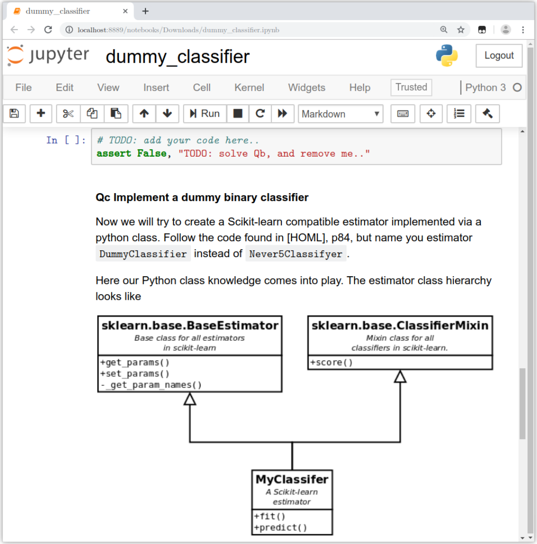
Demo..

Implementing an estimater via a python class as simple as

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

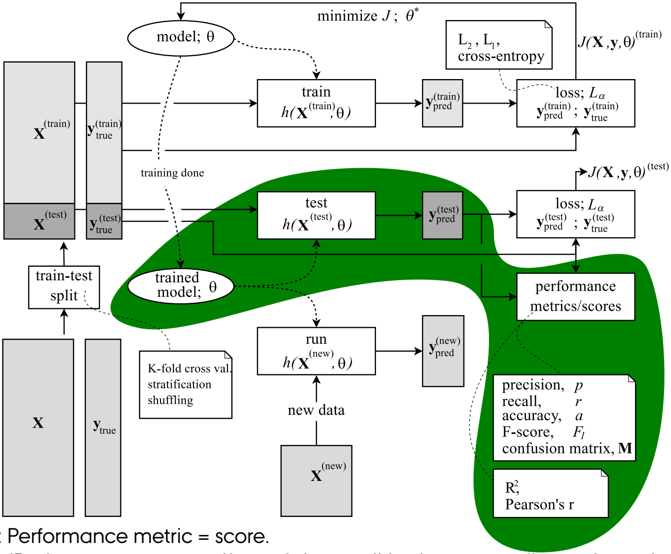
Exercise: L02/dummy_classifier.ipynb

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



Evaluér på test-data: Perfomance metrics

Kort intro til konceptet *performance metrics*..



NOTE$_0$: Performance metric = score.

NOTE$_1$: *'Performance measure'* begreb bruges ikke, kun score eller perf. metric.

NOTE$_2$: Loss er ML algo'ens 'performance mål', score er vores evalueringsmål.

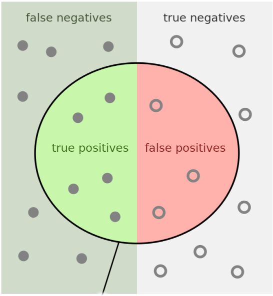
Exercise: `L02/performance_metrics.ipynb`

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

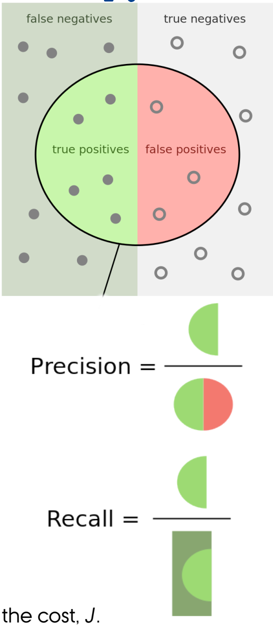
and $N = N_P + N_N$ being the total number of samples and the number of positive and negative samples respectively.



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

Exercise: `L02/performance_metrics.ipynb`

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



$$
\begin{aligned}
\text{precision}, \quad & p &= \frac{TP}{TP+FP} \\
\text{recall (or sensitivity)}, \quad & r &= \frac{TP}{TP+FN} \\
\text{accuracy}, \quad & a &= \frac{TP+TN}{TP+TN+FP+FN} \\
F_1\text{-score}, \quad & F_1 &= \frac{2pr}{p+r}
\end{aligned}
$$

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

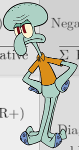
	actual true	actual false
predicted true	TP	FP
predicted false	FN	TN

NOTE$_0$: you can *compare* precision...F_1-score, but not necessarily the cost, J.
NOTE$_1$: beware of matrix transpose and interpretation of 'TP/TN'!

Exercise: `L02/performance_metrics.ipynb`

Nomenclature for the Confusion Matrix

		True condition				
	Total population	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) $= \frac{\text{LR+}}{\text{LR−}}$	F_1 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...

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

Test via the breast cancer Wisconsin dataset...

```python
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_train)
y_pred = clf.predict(X_test)

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

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

NOTE$_0$: for MNIST, a dum classify as '5' $\sim a = 10\%$

NOTE$_1$: 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.

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