

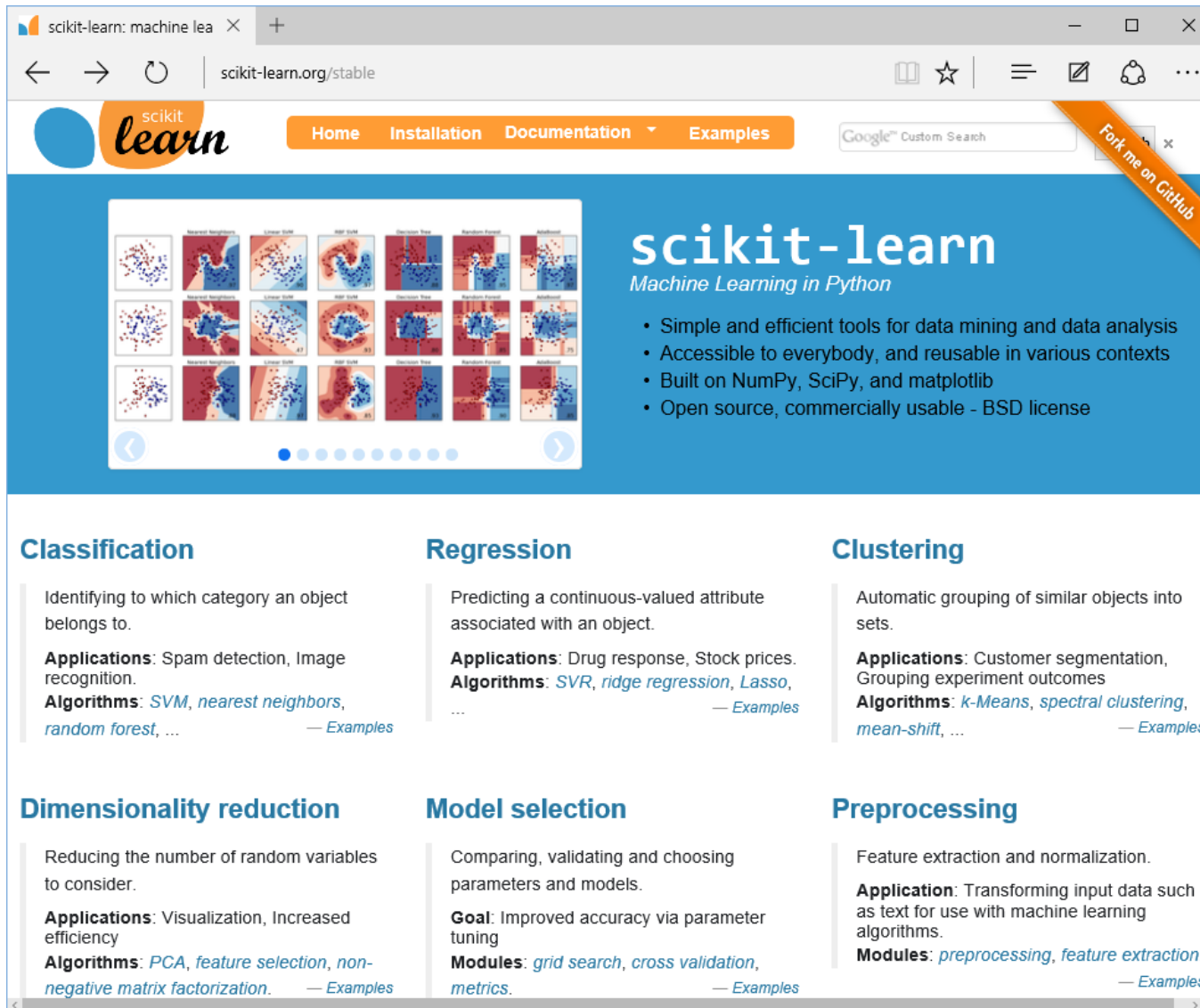
# Machine Learning avec scikit-learn

## Programmation Python

Ricco Rakotomalala

[http://eric.univ-lyon2.fr/~ricco/cours/cours\\_programmation\\_python.html](http://eric.univ-lyon2.fr/~ricco/cours/cours_programmation_python.html)

Scikit-learn est une librairie pour Python spécialisée dans le **machine learning** (apprentissage automatique). Nous utilisons la version **0.19.0** dans ce tutoriel.



The screenshot shows the scikit-learn website. The header includes the scikit-learn logo, navigation links (Home, Installation, Documentation, Examples), a Google Custom Search bar, and a 'Fork me on GitHub' button. The main content area features a grid of 18 small plots illustrating various machine learning concepts. Below the grid, the text 'scikit-learn Machine Learning in Python' is displayed, followed by a list of features: Simple and efficient tools for data mining and data analysis, Accessible to everybody, and reusable in various contexts, Built on NumPy, SciPy, and matplotlib, and Open source, commercially usable - BSD license.

**Classification**

Identifying to which category an object belongs to.

**Applications:** Spam detection, Image recognition.

**Algorithms:** *SVM, nearest neighbors, random forest, ...* — Examples

**Regression**

Predicting a continuous-valued attribute associated with an object.

**Applications:** Drug response, Stock prices.

**Algorithms:** *SVR, ridge regression, Lasso, ...* — Examples

**Clustering**

Automatic grouping of similar objects into sets.

**Applications:** Customer segmentation, Grouping experiment outcomes

**Algorithms:** *k-Means, spectral clustering, mean-shift, ...* — Examples

**Dimensionality reduction**

Reducing the number of random variables to consider.

**Applications:** Visualization, Increased efficiency

**Algorithms:** *PCA, feature selection, non-negative matrix factorization, ...* — Examples

**Model selection**

Comparing, validating and choosing parameters and models.

**Goal:** Improved accuracy via parameter tuning

**Modules:** *grid search, cross validation, metrics, ...* — Examples

**Preprocessing**

Feature extraction and normalization.

**Application:** Transforming input data such as text for use with machine learning algorithms.

**Modules:** *preprocessing, feature extraction, ...* — Examples

**Machine Learning ?** Une discipline de l'informatique (intégrée dans l'intelligence artificielle) destinée à modéliser les relations entre les données. Dans un autre domaine, on parlerait de modélisation statistique, ou de méthodes de data mining, ou encore d'analyse de données.

On retrouve bien – quelle que soit l'appellation utilisée – les grands thèmes du traitement statistique des données [cf. [Introduction à la Data Science](#)]

Impossible de tout traiter dans un seul support. Nous choisissons l'axe de l'**analyse prédictive** en déroulant une étude de cas.

1. Schéma typique d'une démarche d'analyse prédictive
2. Stratégie et évaluation sur les petits échantillons
3. Scoring - Ciblage
4. Recherche des paramètres optimaux des algorithmes
5. Sélection de variables

UCI Machine Learning R

+

←

→


↺

archive.ics.uci.edu/ml/datasets/Pima+Indians+Diab

📖

★

UCI



About

Citation Po

Machine Learning Repository

Center for Machine Learning and Intelligent Systems

Pima Indians Diabetes Data Set

Download: [Data Folder](#), [Data Set Description](#)

Abstract: From National Institute of Diabetes and Digestive and Kidney Diseases; In Peter Turney)

Data Set Characteristics:	Multivariate	Number of Instances:	768	Are
Attribute Characteristics:	Integer, Real	Number of Attributes:	8	Date
Associated Tasks:	Classification	Missing Values?	Yes	Num

Source:  
Original Owners:  
National Institute of Diabetes and Digestive and Kidney Diseases  
Donor of database:  
Vincent Sigillito (vgs '@' aplcen.apl.jhu.edu)  
Research Center, RMI Group Leader  
Applied Physics Laboratory  
The Johns Hopkins University  
Johns Hopkins Road  
Laurel, MD 20707  
(301) 953-6231

Objectif : Prédire / expliquer l'occurrence du diabète (variable à prédire) à partir des caractéristiques des personnes (âge, IMC, etc.) (variables explicatives).  
Fichier texte « pima.txt » avec séparateur « tabulation », première ligne = noms des variables.

pima.txt									
1	pregnant			diastolic		triceps		bodymass	
	pedigree			age plasma		serum		diabete	
2	6	72	35	33.6	0.627	50	148	0	positive
3	1	66	29	26.6	0.351	31	85	0	negative
4	8	64	0	23.3	0.672	32	183	0	positive
5	1	66	23	28.1	0.167	21	89	94	negative
6	0	40	35	43.1	2.288	33	137	168	positive
7	5	74	0	25.6	0.201	30	116	0	negative
8	3	50	32	31	0.248	26	78	88	positive
9	10	0	0	35.3	0.134	29	115	0	negative
10	2	70	45	30.5	0.158	53	197	543	positive
11	8	96	0	0	0.232	54	125	0	positive
12	4	92	0	37.6	0.191	30	110	0	negative
13	10	74	0	38	0.537	34	168	0	positive
14	10	80	0	27.1	1.441	57	139	0	negative
15	1	60	23	30.1	0.398	59	189	846	positive
16	5	72	19	25.8	0.587	51	166	175	positive
17	7	0	0	30	0.484	32	100	0	positive
18	0	84	47	45.8	0.551	31	118	230	positive
19	7	74	0	29.6	0.254	31	107	0	positive
20	1	30	38	43.3	0.183	33	103	83	negative
21	1	70	30	34.6	0.529	32	115	96	positive
22	3	88	41	39.3	0.704	27	126	235	negative
23	8	84	0	35.4	0.388	50	99	0	negative

Ln : 1

Col : 1

Sel : 0 | 0

Dos\Windows

UTF-8

INS

Schéma typique (standard) de l'analyse prédictive

# **ANALYSE PRÉDICTIVE**

Y : variable cible (diabète)

X1, X2, ... : variables explicatives

f(.) une fonction qui essaie d'établir la relation  $Y = f(X1, X2, ...)$

f(.) doit être « aussi précise que possible »...

Ensemble  
d'apprentissage

Construction de la fonction f(.) à  
partir des données d'apprentissage

age	sex	height	weight	pedigree	skin	insulin	diabetes
4	72	20.126	0.437	50	148	0	positive
1	66	29.266	0.351	31	86	0	negative
8	66	0.233	0.672	32	183	0	positive
1	66	23.381	0.187	21	89	0	negative
0	40	36.431	2.208	33	137	160	positive
5	74	0.256	0.201	30	116	0	negative
3	50	32	31.0.249	26	78	80	positive
10	0	0.353	0.134	29	115	0	negative
2	70	45.305	0.159	53	197	543	positive
9	96	0	0.232	54	125	0	positive
4	92	0.376	0.191	30	110	0	negative
10	74	0	30.0.637	34	168	0	positive
10	80	0.271	1.441	57	139	0	negative
1	60	23.301	0.389	59	189	840	positive
5	72	18.250	0.607	61	166	175	positive
7	0	0	30.0.484	32	100	0	positive
0	64	47.466	0.561	31	118	220	positive
7	74	0.296	0.254	31	107	0	positive
1	30	39.433	0.183	33	101	83	negative
1	70	30.346	0.529	32	115	96	positive
3	88	41.393	0.704	27	126	235	negative
8	64	0.354	0.389	50	99	0	negative
7	90	0.398	0.451	41	196	0	positive
9	60	35	29.0.563	29	119	0	positive
11	94	33.366	0.254	51	143	146	positive
10	70	26.311	0.295	41	125	115	positive
7	76	0.394	0.257	43	147	0	positive
1	66	15.232	0.487	22	97	140	negative
13	82	18.222	0.145	57	146	110	negative
5	82	0.341	0.337	38	117	0	negative

Ensemble  
de données  
(dataset)

Ensemble de test

$$Y = f(X1, X2, ...) + \epsilon$$

Application du modèle (prédiction)  
sur l'ensemble de test

$$(Y, \hat{Y})$$

Y : valeurs observées  
Y^ : valeurs prédites par f(.)

Mesures de **performances**  
par confrontation entre Y  
et Y^ : matrice de  
confusion + mesures

Pandas : Python Data Analysis Library. Très pratique pour la manipulation des données, avec un type data.frame inspiré de R.

#utilisation de la librairie Pandas

#spécialisée - entres autres - dans la manipulation des données

import pandas

pima = pandas.read\_table("pima.txt", sep="\t", header=0)

header = 0, la première ligne (n°0) correspond aux noms de variables.

#dimensions

print(pima.shape) # (768, 9) 768 lignes (obs.) et 9 colonnes (variables)

#liste des colonnes

print(pima.columns) # Index(['pregnant', 'diastolic', 'triceps', 'bodymass', 'pedigree', 'age', 'plasma', 'serum', 'diabete'], dtype='object')

#liste des colonnes et leurs types

print(pima.dtypes)



pregnant	int64
diastolic	int64
triceps	int64
bodymass	float64
pedigree	float64
age	int64
plasma	int64
serum	int64
diabete	object
dtype: object (chaîne de caractères)	

```
#transformation en matrice numpy - seul reconnu par scikit-learn
```

```
data = pima.values
```

```
#X matrice des var. explicatives
```

```
X = data[:,0:8]
```

```
#y vecteur de la var. à prédire
```

```
y = data[:,8]
```

```
#utilisation du module model_selection de scikit-learn (sklearn)
```

```
from sklearn import model_selection
```

```
#subdivision des données – éch.test = 300 ; éch.app = 768 – éch.test = 468
```

```
X_app,X_test,y_app,y_test = model_selection.train_test_split(X,y,test_size = 300,random_state=0)
```

```
print(X_app.shape,X_test.shape,y_app.shape,y_test.shape)
```

(468,8)

(300,8)

(468,)

(300,)



```
#à partir du module linear_model du package sklearn
#importer la classe LogisticRegression
from sklearn.linear_model import LogisticRegression


#création d'une instance de la classe
lr = LogisticRegression(solver="liblinear")

#exécution de l'instance sur les données d'apprentissage
#c.à-d. construction du modèle prédictif
modele = lr.fit(X_app,y_app)

#les sorties sont très pauvres à la différence des logiciels de stat
#les coefficients...
print(modele.coef_,modele.intercept_)
```

Nous utilisons la [régression logistique](#), très populaire en France. D'autres [approches supervisées](#) sont disponibles dans scikit-learn.

On ne dispose pas des indicateurs usuels de la régression logistique (tests de significativité, écarts-type des coefficients, etc.)



```
[[ 8.75111754e-02 -1.59515113e-02  1.70447729e-03  5.18540256e-02
  5.34746050e-01  1.24326526e-02  2.40105095e-02 -2.91593120e-04]] [-5.13484535]
```

Attention : La régression logistique de [scikit-learn](#) s'appuie sur un algorithme différent de celui des logiciels de statistique.

Coefficients du logiciel SAS

Variable	Coefficient
Intercept	8.4047
pregnant	-0.1232
diastolic	0.0133
triceps	-0.0006
bodymass	-0.0897
pedigree	-0.9452
age	-0.0149
plasma	-0.0352
serum	0.0012

Coefficients du modèle élaboré sur la totalité des données (scikit-learn)

Variable	Coefficient
Intercept	5.8844
pregnant	-0.1171
diastolic	0.0169
triceps	-0.0008
bodymass	-0.0597
pedigree	-0.6776
age	-0.0072
plasma	-0.0284
serum	0.0006

Les coefficients sont du même ordre mais différents. Ca ne veut pas dire que le modèle est moins performant en prédiction.



sklearn.linear\_model.LogisticRegression

```
class sklearn.linear_model.LogisticRegression(penalty='l2', dual=False,
tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None,
random_state=None, solver='liblinear', max_iter=100, multi_class='ovr', verbose=0)
```

[source]

Logistic Regression (aka logit, MaxEnt) classifier.

In the multiclass case, the training algorithm uses the one-vs-rest (OvR) scheme if the 'multi\_class' option is set to 'ovr' and uses the cross-entropy loss, if the 'multi\_class' option is set to 'multinomial'. (Currently the 'multinomial' option is supported only by the 'lbfgs' and 'newton-cg' solvers.)

This class implements regularized logistic regression using the liblinear library, newton-cg and lbfgs solvers. It can handle both dense and sparse input. Use C-ordered arrays or CSR matrices containing 64-bit floats for optimal performance; any other input format will be converted (and copied).

The newton-cg and lbfgs solvers support only L2 regularization with primal formulation. The liblinear solver supports both L1 and L2 regularization, with a dual formulation only for the L2 penalty.

```
#prediction sur l'échantillon test
```

```
y_pred = modele.predict(X_test)
```

```
#importation de metrics - utilisé pour les mesures de performances
```

```
from sklearn import metrics
```

```
#matrice de confusion
```

```
#confrontation entre Y obs. sur l'éch. test et la prédiction
```

```
cm = metrics.confusion_matrix(y_test,y_pred)
```

```
print(cm)
```

```
#taux de succès
```

```
acc = metrics.accuracy_score(y_test,y_pred)
```

```
print(acc) # 0.793 = (184 + 54) / (184 + 17 + 45 + 54)
```

```
#taux d'erreur
```

```
err = 1.0 - acc
```

```
print(err) # 0.206 = 1.0 - 0.793
```

```
#sensibilité (ou rappel)
```

```
se = metrics.recall_score(y_test,y_pred,pos_label='positive')
```

```
print(se) # 0.545 = 54 / (45 + 54)
```

**Matrice de confusion**

Ligne : observé

Colonne : prédiction

	184	17
	45	54

#écrire sa propre func. d'éval - ex. spécificité

**def** specificity(y,y\_hat):

#matrice de confusion – un objet **numpy.ndarray**

mc = metrics.confusion\_matrix(y,y\_hat)

#"negative" est sur l'indice 0 dans la matrice

**import** numpy

res = mc[0,0]/numpy.sum(mc[0,:])

#retour

**return** res

#

#la rendre utilisable - transformation en objet scorer

**specificite** = metrics.make\_scorer(specificity,greater\_is\_better=True)

#utilisation de l'objet scorer

#remarque : modele est le modèle élaboré sur l'éch. d'apprentissage

sp = **specificite**(modele,X\_test,y\_test)

print(sp) # 0.915 = 184 / (184 + 17)

Remarque : utiliser les packages comme simples boîtes à outils est une chose, savoir programmer en Python en est une autre. C'est indispensable si l'on veut aller plus loin et exploiter au mieux le dispositif.

Matrice de confusion =

[[184	17]
[ 45	54]]

Stratégie et évaluation sur les petits échantillons

# **VALIDATION CROISÉE**

```
#importer la classe LogisticRegression
from sklearn.linear_model import LogisticRegression

#création d'une instance de la classe
lr = LogisticRegression(solver="liblinear")

#exécution de l'instance sur la totalité des données (X,y)
modele_all = lr.fit(X,y)
```

```
#affichage
print(modele_all.coef_,modele_all.intercept_)
# [[ 1.17056955e-01 -1.69020125e-02  7.53362852e-04  5.96780492e-02  6.77559538e-01  7.21222074e-03  2.83668010e-02 -6.41169185e-04]] [-5.8844014]
# !!! Les coefficients sont différents de ceux estimés sur l'éch. d'apprentissage (on a plus d'obs. ici) !!!
```

```
#utilisation du module model_selection
from sklearn import model_selection

#évaluation en validation croisée : 10 cross-validation
succes = model_selection.cross_val_score(lr,X,y,cv=10,scoring='accuracy')
```

```
#détail des itérations
print(succes)
```

```
#moyenne des taux de succès = estimation du taux de succès en CV
print(succes.mean()) # 0.767
```

Problème : lorsque l'on traite un petit fichier (en nombre d'obs.), le schéma apprentissage – test est pénalisant (apprentissage : on réduit l'information disponible pour créer le modèle ; test : un faible effectif produit des estimations des performances très instables).

Solution : (1) construire le modèle sur la totalité des données, (2) évaluer les performances à l'aide des techniques de ré-échantillonnage (ex. validation croisée)

```
0.74025974
0.75324675
0.79220779
0.72727273
0.74025974
0.74025974
0.81818182
0.79220779
0.73684211
0.82894737
```

Dans un schéma apprentissage - test

**SCORING**

Ex. de ciblage : faire la promotion d'un produit auprès d'un ensemble de clients

Objectif : contacter le moins de personnes possible, obtenir le max. d'achats

Démarche : attribuer un score aux individus, les trier de manière décroissante (score élevé = forte appétence au produit), estimer à l'aide de la courbe de gain le nombre d'achats en fonction d'une taille de cible choisie.

Remarque : L'idée peut être transposée à d'autres domaines (ex. dépistage de maladie)

Ensemble  
d'apprentissage

Construction de la fonction  $f(.)$  à  
partir des données d'apprentissage

age	sex	status	income	education	occupation	score	target
6	72	35	13.6	0.627	50	148	0 positive
1	66	29	26.6	0.351	31	86	0 negative
8	64	0	23.3	0.472	32	103	0 positive
1	66	23	28.1	0.187	21	89	0 negative
0	40	38	43.1	2.288	33	137	100 positive
5	74	0	25.6	0.201	30	116	0 negative
3	50	32	31.0	2.468	26	78	89 positive
10	0	0	35.3	0.134	29	115	0 negative
2	70	45	30.5	0.120	53	197	543 positive
6	96	0	0.0	2.232	54	125	0 positive
4	92	0	37.6	0.191	30	110	0 negative
10	74	0	36.0	5.537	34	160	0 positive
10	80	0	27.1	1.441	57	139	0 negative
1	60	23	30.1	0.386	59	109	186 positive
5	72	19	25.8	0.587	51	166	175 positive
7	0	0	30.0	4.644	32	100	0 positive
0	84	47	45.8	0.551	31	118	239 positive
7	74	0	29.6	0.254	31	107	0 positive
1	30	38	43.3	0.183	33	103	63 negative
1	70	30	34.6	0.539	32	115	96 positive
3	88	41	39.3	0.704	27	126	225 negative
9	84	0	35.4	0.388	50	99	0 negative
7	90	0	39.6	0.451	41	105	0 positive
9	80	35	29.0	2.632	29	119	0 positive
11	94	33	36.6	0.254	51	143	148 positive
10	70	26	31.1	0.205	41	125	115 positive
7	76	0	39.4	0.257	43	147	0 positive
1	66	15	23.2	0.487	22	97	140 negative
13	82	19	22.2	0.245	57	145	110 negative
5	92	0	34.1	0.337	38	117	0 negative

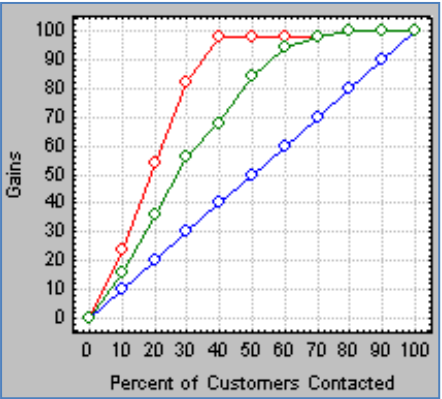
$$Y = f(X_1, X_2, \dots) + \varepsilon$$

Application du modèle (scoring)  
sur l'ensemble de test

$(Y, score)$

Y : valeurs observées  
score : propension à être positif  
attribué par  $f(.)$

Mesurer la performance  
avec la courbe de gain



Ensemble  
de données  
(dataset)

Ensemble de test



#classe Régression Logistique

from sklearn.linear\_model import LogisticRegression

#création d'une instance de la classe

lr = LogisticRegression(solver="liblinear")

#modélisation sur les données d'apprentissage

modele = lr.fit(X\_app,y\_app)

#calcul des probas d'affectation sur ech. test

probas = lr.predict\_proba(X\_test)

#score de 'presence'

score = probas[:,1] # [0.86238322 0.21334963 0.15895063 ...]

#transf. en 0/1 de Y\_test

pos = pandas.get\_dummies(y\_test).values

#on ne récupère que la 2è colonne (indice 1)

pos = pos[:,1] # [1 0 0 1 0 0 1 1 ...]

#nombre total de positif

import numpy

npos = numpy.sum(pos) # 99 – il y a 99 ind. “positifs” dans l'échantillon test

Probas. d'affectation aux classes  
Negative, Positive

```
[ 0.13761678, 0.86238322],
[ 0.78665037, 0.21334963],
[ 0.84104937, 0.15895063],
[ 0.39960826, 0.60039174],
[ 0.81646421, 0.18353579],
[ 0.91705129, 0.08294871],
[ 0.32575719, 0.67424281],
[ 0.27436772, 0.72563228],
[ 0.56763049, 0.43236951],
```

Classe d'appartenance  
Negative, Positive

```
[ 0., 1.],
[ 1., 0.],
[ 1., 0.],
[ 0., 1.],
[ 1., 0.],
[ 1., 0.],
[ 0., 1.],
[ 0., 1.],
```

L'individu n°55 a le score le plus faible, suivi du n°45, ... , l'individu n°159 a le score le plus élevé.

#index pour tri selon le score croissant

```
index = numpy.argsort(score) # [ 55  45 265 261 ... 11 255 159]
```

#inverser pour score décroissant – on s'intéresse à forte proba. en priorité

```
index = index[::-1] # [ 159 255 11 ... 261 265 45 55]
```

#tri des individus (des valeurs 0/1)

```
sort_pos = pos[index] # [ 1  1  1  1  1  0  1  1 ...]
```

Le score est pas trop mal, il place bien les 'positifs' en tête des données après le tri décroissant.

#somme cumulée

```
cpos = numpy.cumsum(sort_pos) # [ 1  2  3  4  5  5  6  7 ... 99]
```

#rappel

```
rappel = cpos/npos # [ 1/99 2/99 3/99 4/99 5/99 5/99 6/99 7/99 ... 99/99]
```

#nb. obs ech.test

```
n = y_test.shape[0] # 300, il y a 300 ind. dans l'éch. test
```

#taille de cible – séquence de valeurs de 1 à 300 avec un pas de 1

```
taille = numpy.arange(start=1,stop=301,step=1) # [1  2  3  4  5 ... 300]
```

#passer en proportion

```
taille = taille / n # [ 1/300 2/300 3/300 ... 300/300 ]
```

```
#graphique avec matplotlib  
import matplotlib.pyplot as plt
```

```
#titre et en-têtes
```

```
plt.title('Courbe de gain')  
plt.xlabel('Taille de cible')  
plt.ylabel('Rappel')
```

```
#limites en abscisse et ordonnée
```

```
plt.xlim(0,1)  
plt.ylim(0,1)
```

```
#astuce pour tracer la diagonale
```

```
plt.scatter(taille, taille, marker='.', color='blue')
```

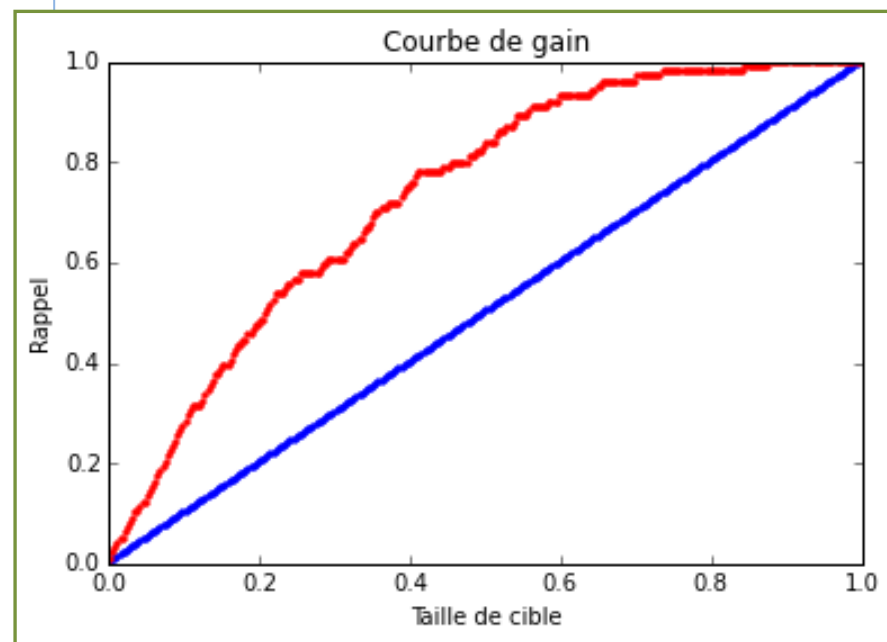
```
#insertion du couple (taille, rappel)
```

```
plt.scatter(taille, rappel, marker='.', color='red')
```

```
#affichage
```

```
plt.show()
```

Objectif : Réaliser le graphique à partir des vecteurs 'taille' et 'rappel' – On parle de « courbe de gain » ou « courbe lift cumulée ».



Recherche des paramètres optimaux des algorithmes

# GRID SEARCH

```
#svm
```

```
from sklearn import svm
```

```
#par défaut un noyau RBF et C = 1.0
```

```
mvs = svm.SVC()
```

```
#modélisation
```

```
modele2 = mvs.fit(X_app,y_app)
```

```
#prédiction ech. test
```

```
y_pred2 = modele2.predict(X_test)
```

```
#matrice de confusion
```

```
print(metrics.confusion_matrix(y_test,y_pred2))
```

```
#succès en test
```

```
print(metrics.accuracy_score(y_test,y_pred2)) # 0.67
```

Problème : De nombreux algorithmes de machine learning reposent sur des paramètres qui ne sont pas toujours évidents à déterminer pour obtenir les meilleures performances sur un jeu de données à traiter. Ex. **SVM**.

```
class sklearn.svm.SVC(C=1.0, kernel='rbf', degree=3, gamma=0.0,
coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200,
class_weight=None, verbose=False, max_iter=-1, random_state=None)
```

La méthode ne fait pas mieux que le classifieur par défaut (prédire systématiquement 'negative', classe majoritaire). Matrice de confusion :

[[201	0]
[ 99	0]]

C'est la méthode (SVM) qui est inapte ou c'est le paramétrage qui est inadapté ?

```
#import de la classe
from sklearn import model_selection

#combinaisons de paramètres à évaluer
parametres = [{'C':[0.1,1,10],'kernel':['rbf','linear']}]

#évaluation en validation croisée de 3 x 2 = 6 configurations
#accuracy sera le critère à utiliser pour sélectionner la meilleure config
#mvs est l'instance de la classe svm.SVC (cf. page précédente)
grid = model_selection.GridSearchCV(estimator=mvs,param_grid=parametres,scoring='accuracy')

#lancer la recherche – attention, gourmand en calculs
grille = grid.fit(X_app,y_app)

#résultat pour chaque combinaison
print(pandas.DataFrame.from_dict(grille.cv_results_).loc[:,["params","mean_test_score"]])

#meilleur paramétrage
print(grille.best_params_) # {'C' : 10, 'kernel' : 'linear'}

#meilleure performance – estimée en interne par validation croisée
print(grille.best_score_) # 0.7564

#prédiction avec le modèle « optimal » c.-à-d. {'C' : 10, 'kernel' : 'linear'}
y_pred3 = grille.predict(X_test)

#taux de succès en test
print(metrics.accuracy_score(y_test,y_pred3)) # 0.7833, on se rapproche de la rég. logistique
```

Stratégie : Grille de recherche. On indique les paramètres à faire varier, **scikit-learn** les croise et mesure les performances en validation croisée.

	params	mean_test_score
0	{'C': 0.1, 'kernel': 'rbf'}	0.638889
1	{'C': 0.1, 'kernel': 'linear'}	0.752137
2	{'C': 1, 'kernel': 'rbf'}	0.638889
3	{'C': 1, 'kernel': 'linear'}	0.747863
4	{'C': 10, 'kernel': 'rbf'}	0.638889
5	{'C': 10, 'kernel': 'linear'}	0.756410

Réduire le modèle aux variables explicatives les plus pertinentes

## **SÉLECTION DE VARIABLES**

Liste initiale des variables : pregnant, diastolic, triceps, bodymass, pedigree, age, plasma, serum.

```
#importer la classe LogisticRegression
from sklearn.linear_model import LogisticRegression
```

```
#création d'une instance de la classe
lr = LogisticRegression(solver="liblinear")
```

```
#algorithme de sélection de var.
from sklearn.feature_selection import RFE
selecteur = RFE(estimator=lr)
```

```
#lancer la recherche
sol = selecteur.fit(X_app, y_app)
```

```
#nombre de var. sélectionnées
print(sol.n_features_) # 4 → 4 = 8 / 2 variables sélectionnées
```

```
#liste des variables sélectionnées
print(sol.support_) # [True False False True True False True False]
```

```
#ordre de suppression
print(sol.ranking_) # [1 2 4 1 1 3 1 5]
```

Objectif : la sélection de variables - la recherche de modèles parcimonieux - présente plusieurs avantages : interprétation, déploiement (moins de var. à renseigner), performances en généralisation (ou du moins maintien des performances).

Méthode : Nous implémentons la méthode [RFE](#) de scikit-learn : elle élimine au fur et à mesure les coefficients les plus faibles en valeur absolue (étrange : les variables ne sont pas toujours à la même échelle ??? une standardisation des variables me paraît nécessaire), et s'arrête quand on arrive à la moitié ou à un nombre spécifié de variables.

Variables sélectionnées :  
pregnant, bodymass, pedigree, plasma.

**Serum** a été retirée en premier, puis **triceps**, puis **âge**, puis **diastolic**. Les variables restantes sont indexées **1**.



```
#réduction de la base d'app. aux var. sélectionnées
#en utilisant le filtre booléen sol.support_
X_new_app = X_app[:,sol.support_]
print(X_new_app.shape) # (468, 4) → 4 variables restantes

#construction du modèle sur les explicatives sélectionnées
modele_sel = lr.fit(X_new_app,y_app)

#réduction de la base test aux mêmes variables
X_new_test = X_test[:,sol.support_]
print(X_new_test.shape) # (300, 4)

#prédiction du modèle réduit sur l'éch. test
y_pred_sel = modele_sel.predict(X_new_test)

#évaluation
print(metrics.accuracy_score(y_test,y_pred_sel)) # 0.787
```

Aussi bien (presque, 0.793)  
que le modèle initial, mais  
avec moitié moins de  
variables.

De la documentation à profusion (n'achetez pas des livres sur Python)

Site du cours

[http://eric.univ-lyon2.fr/~ricco/cours/cours\\_programmation\\_python.html](http://eric.univ-lyon2.fr/~ricco/cours/cours_programmation_python.html)

Site de Python

Welcome to Python - <https://www.python.org/>

Python 3.4.3 documentation - <https://docs.python.org/3/index.html>

Portail Python

Page Python de [Developpez.com](http://developpez.com)

Quelques cours en ligne

P. Fuchs, P. Poulain, « [Cours de Python](#) » sur Developpez.com

G. Swinnen, « [Apprendre à programmer avec Python](#) » sur Developpez.com

« [Python](#) », Cours interactif sur [Codecademy](#)

POLLS (KDNuggets)

**Data Mining / Analytics Tools Used**

Python, 4<sup>ème</sup> en [2015](#)

**What languages you used for data mining / data science?**

Python, 3<sup>ème</sup> en [2014](#) (derrière R et SAS)