Machine Learning with scikit-learn

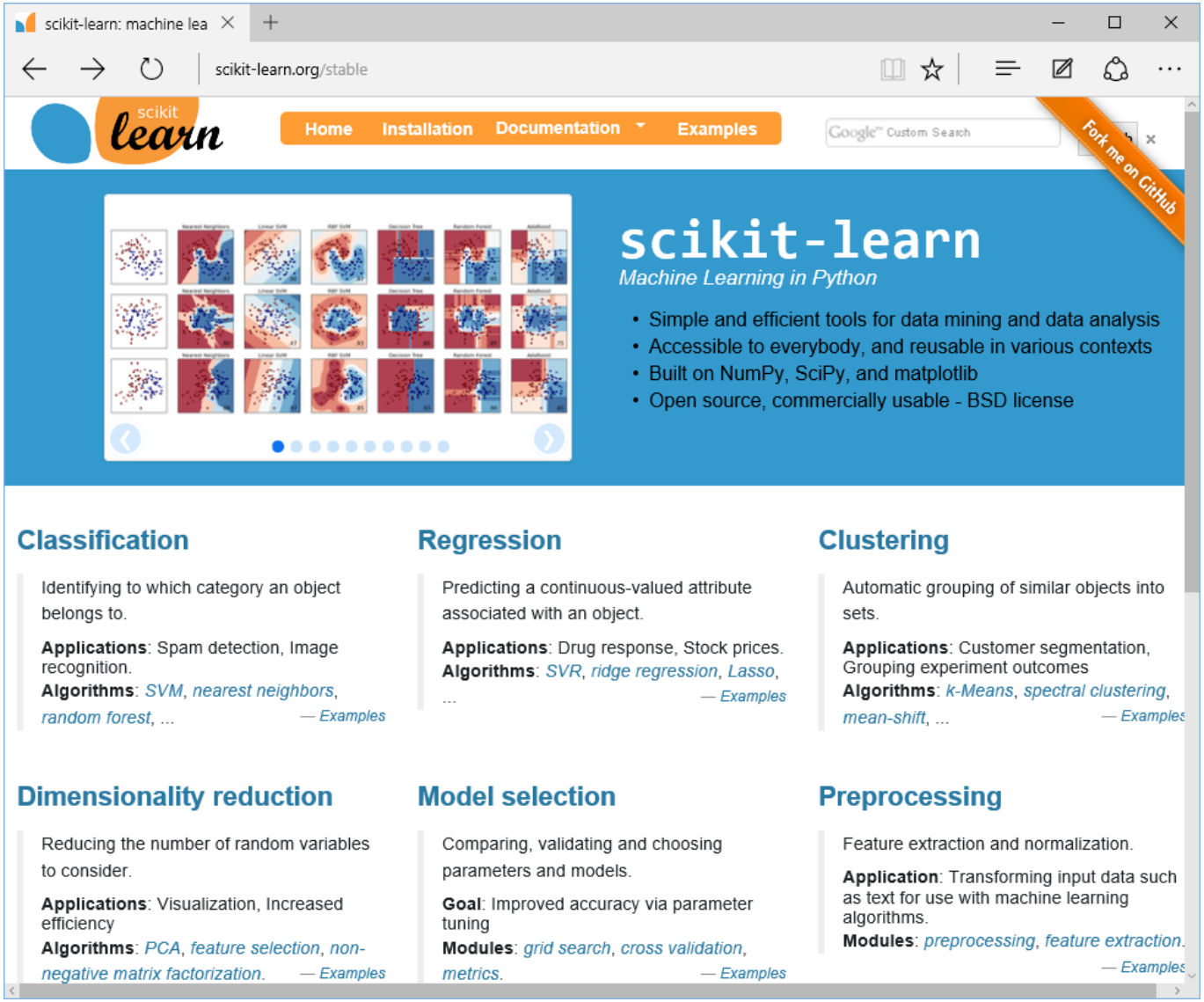
Python Programming

Ricco Rakotomalala

http://eric.univ-lyon2.fr/~ricco/cours/cours_programmation_python.html

Scikit-learn?

Scikit-learn is a package for performing machine learning in Python. It incorporates various algorithms for classification, regression, clustering, etc. We use 0.19.0 in this tutorial.

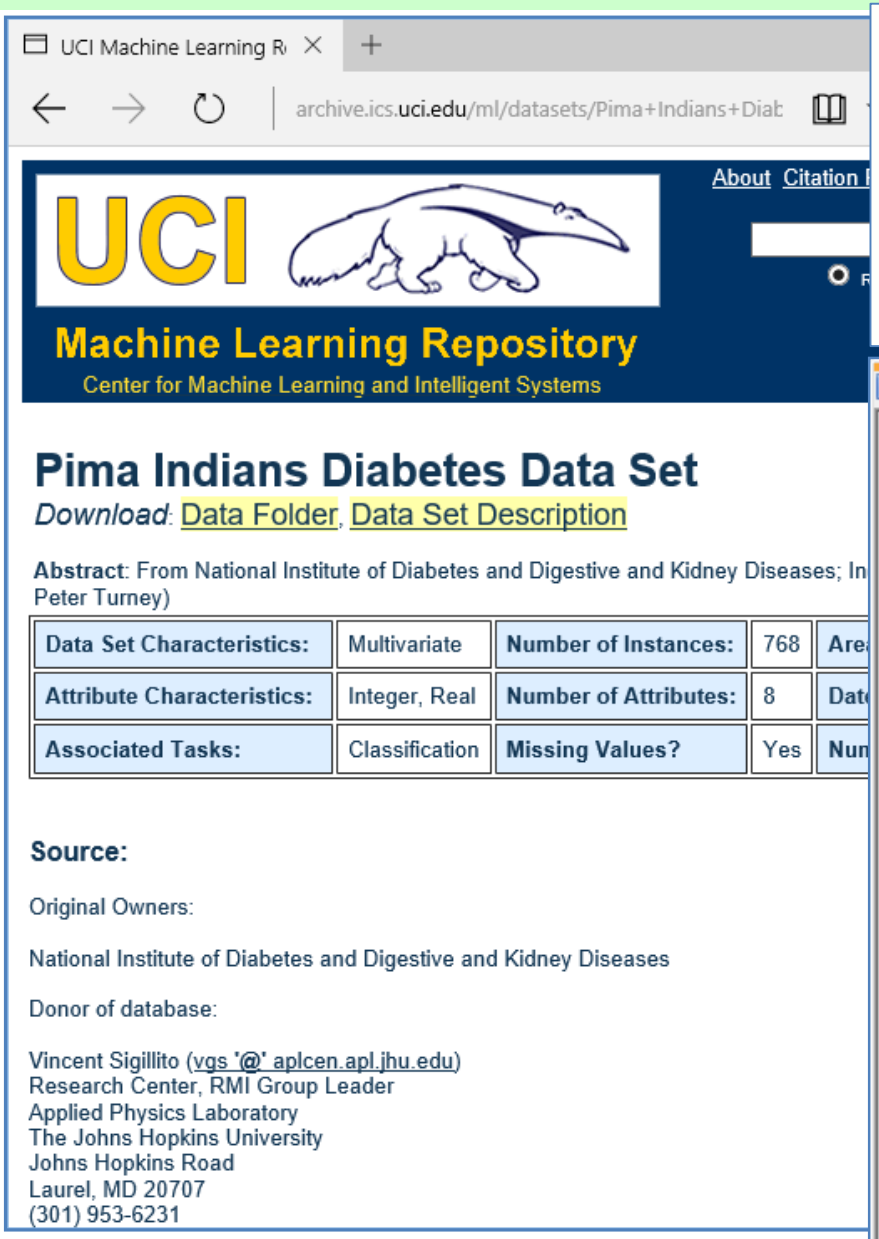


Machine learning explores the study and construction of algorithms that can learn from and make predictions on data. Such algorithms operate by building a model from example inputs in order to make data-driven predictions or decisions…. Machine learning is closely related to computational statistics; a discipline that aims at the design of algorithms for implementing statistical methods on computers (Wikipedia).

We cannot to treat all the features of sckit-learn in one slideshow.
We focus on the classification problem here.

1. A typical classification process
2. Cross-validation evaluation for small dataset
3. Scoring process – Gains chart
4. Search for optimal parameters for algorithms
5. Feature selection

Dataset – PIMA INDIAN DIABETES



Goal: Predict / explain the occurrence of diabetes (target variable) from the characteristics of individuals (age, BMI, etc.) (descriptors). The « pima.txt » data file is in the TSV (tab-separated values) text format (first row = attributes name).

pima.txt

	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

A typical classification process

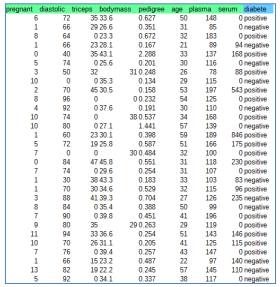
CLASSIFICATION PROCESS

Y : target attribute (diabete)

X1, X2, … : predictive attributes

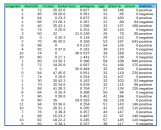
f(.) the underlying concept with Y = f(X1, X2, …)

f(.) must be "as accurate as possible" …

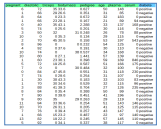


Dataset

Training set



Test set



Learning the function f(.) (the parameters of the function) from the training set

$Y = f(X1,X2,\ldots) + \varepsilon$

Classification of the test set i.e. the model is applied on the test set to obtain the predicted values

$(Y, \hat{Y})$

Measuring the accuracy of the prediction by comparing Y and Y^: confusion matrix and evaluation measurements

Y : observed class values

Y^ : predicted class values from f(.)

Pandas: Python Data Analysis Library. The package Pandas provides useful tools for handling, among others, flat data file. A R-like "data frame" structure is available.

```
#import the Pandas library
import pandas
pima = pandas.read_table("pima.txt",sep="\t",header=0)

#number of rows and columns
print(pima.shape) # (768, 9)

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

#data type for each column
print(pima.dtypes)
```

header = 0, the first row (n°0) correspond to the columns name

768 rows (instances) and 9 columns (attributes)

```
pregnant      int64
diastolic     int64
triceps       int64
bodymass    float64
pedigree    float64
age           int64
plasma        int64
serum         int64
diabete      object
dtype: object (string for our dataset)
```

```
#transform the data into a NumPy matrix
data = pima.as_matrix()

#X matrix for the descriptors (input attributes)
X = data[:,0:8]

#y vector for the target attribute
y = data[:,8]

#using the model_selection module of scikit-learn (sklearn)
from sklearn import model_selection

#test set size = 300 ; training set size = 768 – test set = 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,)

We use the logistic regression. Many supervised learning methods are available in scikit-learn.

```
#from the linear_model module of sklearn
#import the LogisticRegression class
from sklearn.linear_model import LogisticRegression

#lr is an object from the LogisticRegression class
lr = LogisticRegression()

#fitting the model to the labelled training set
#X_app: input data, y_app: target attribute (labels)
modele = lr.fit(X_app,y_app)

#the outputs are lacking
#the coefficients and the intercept
print(modele.coef_,modele.intercept_)
```

```
[[ 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]
```

There are not the usual outputs for logistic regression (tests of significance, standard error of the coefficients, etc.)

Note: The logistic regression of scikit-learn is based on other algorithm than the state-of-art ones (e.g. SAS proc logistic or R glm algorithms)

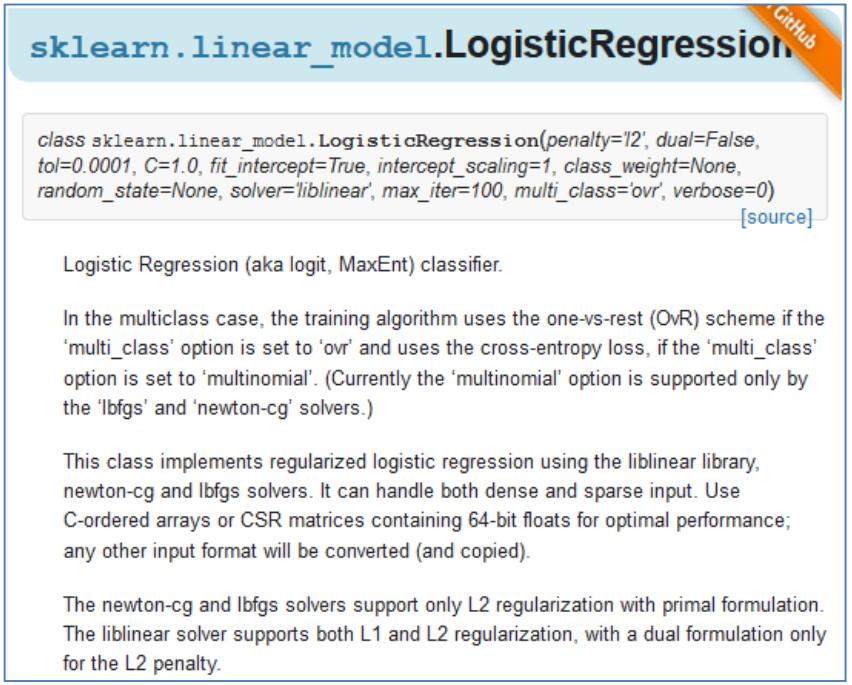
Coefficients of SAS

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 of 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

The coefficients are similar but different. It does not mean that the model is less efficient in prediction.



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 and evaluation on the test set

```
#prediction on the test sample
y_pred = modele.predict(X_test)

#metrics – quantifying the quality of the prediction
from sklearn import metrics

#confusion matrix
#comparison of the observed target values and the prediction
cm = metrics.confusion_matrix(y_test,y_pred)
print(cm)

#accuracy rate
acc = metrics.accuracy_score(y_test,y_pred)
print(acc) # 0.793 = (184 + 54)/ (184 + 17 + 45 + 54)

#error rate
err = 1.0 - acc
print(err) # 0.206 = 1.0 – 0.793

#recall (sensibility)
se = metrics.recall_score(y_test,y_pred,pos_label='positive')
print(se) # 0.545 = 54 / (45+ 54)
```

Confusion matrix
Row: observed
Column: prediction

```
[[184  17]
 [ 45  54]]
```

```
#a function for computing specificity
def specificity(y,y_hat):
    #confusion matrix – a numpy.ndarray object
    mc = metrics.confusion_matrix(y,y_hat)
    #"negative" is the first row (index 0) of the matrix
    import numpy
    res = mc[0,0]/numpy.sum(mc[0,:])
    #return the specificity
    return res
#

# make the function usable as a scorer object
specificite = metrics.make_scorer(specificity,greater_is_better=True)

#using the new scorer object
#modele is the classifier fitted on the training set (see page 9)
sp = specificite(modele,X_test,y_test)
print(sp) # 0.915 = 184 / (184 + 17)
```

Note: Use the package like a simple toolbox is one thing, programming in Python is another. This skill is essential if we want to go further.

Confusion matrix =

```
[[184  17]
 [ 45  54]]
```

Measuring performance on small dataset

CROSS VALIDATION

Issue: When dealing with a small file, the subdivision of data into learning and test samples is penalizing. Indeed, we will have less instances to build an effective model, and the estimate of the error will be unreliable because based on too few observations.

Solution: (1) Learning the classifier using the whole dataset. (2) Evaluate the performance of this classifier using the cross-validation mechanism.

```
#import the LogisticRegression class
from sklearn.linear_model import LogisticRegression

#instantiate and initialize the object
lr = LogisticRegression()

#fit on the whole dataset (X,y)
modele_all = lr.fit(X,y)

#print the coefficients and the intercept
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]
# !!! Of course, the coefficients and the intercept are not the same as the ones estimated on the training set !!!

#import the model_selection module
from sklearn import model_selection

#10-fold cross-validation to evaluate the success rate
succes = model_selection.cross_val_score(lr,X,y,cv=10,scoring='accuracy')

#details of the results for each fold
print(succes)

#mean of the success rate = cross-validation estimation of the success rate of modele_all
print(succes.mean()) # 0.767
```

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

Gains chart

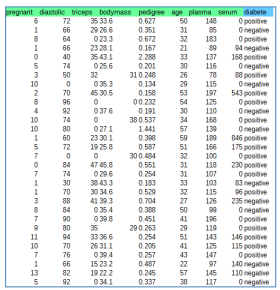
SCORING

Ex. of direct marketing: identify the likely responders to a mailing (1)

Goal: contact the fewest people, get the max of purchases

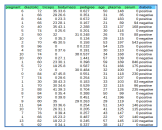
Process: assign a "probability of responding" score to individuals, sort them in a decreasing way (high score = high probability to purchase), estimate the number of purchases for a given target size (number of customer to contact) using the gain chart

Note: The idea can be transposed to other areas (e.g. disease screening)

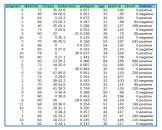


Dataset

Training set



Test set



Construction of the classifier f(.) which can calculate the probability (or any value proportional to the probability) on an instance to be positive (the class of interest)

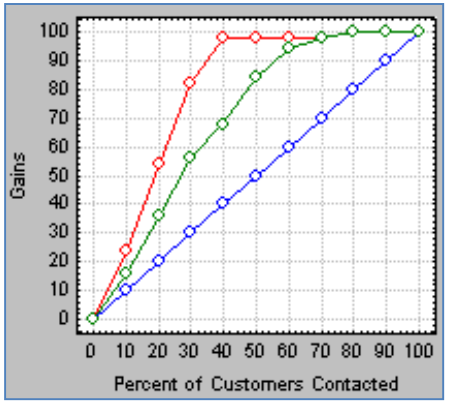
$Y = f(X1,X2,\ldots) + \varepsilon$

Calculate the score of instances in the test set

$(Y, score)$

Y : observed class values
score : probability of responding computed by f(.)

Measuring the performance using the gain chart



```
#Logistic Regression class
from sklearn.linear_model import LogisticRegression
#instantiate and initialize the object
lr = LogisticRegression()
#fit the model to the training sample
modele = lr.fit(X_app,y_app)
#calculate the posterior probabilities for the test sample
probas = lr.predict_proba(X_test)
#score for 'presence' (positive class value)
score = probas[:,1] # [0.86238322  0.21334963  0.15895063 …]
#transforming in 0/1 (dummy variables) the Y_test vector
pos = pandas.get_dummies(y_test).as_matrix()
#get the second column (index = 1)
pos = pos[:,1] # [ 1  0  0  1  0  0  1  1 …]
#number of "positive" instances
import numpy
npos = numpy.sum(pos) # 99 – there are 99 ''positive" instances into the test set
```

Class membership probabilities
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],
```

Class membership
Negative, Positive

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

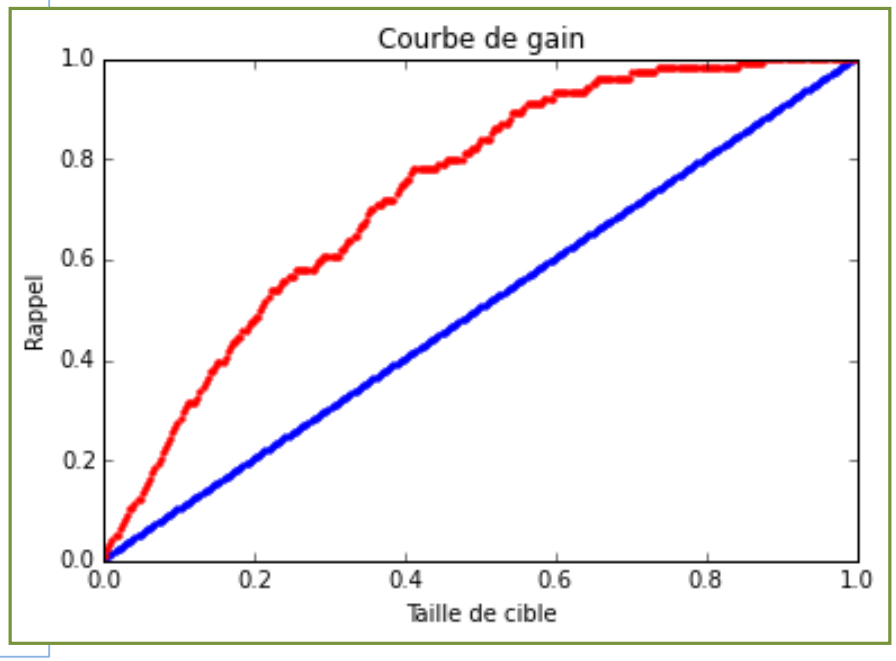
The individual n°55 has the lowest score, then the n°45, … , the individual n°159 has the highest score.

```
#indices that would sort according to the score
index = numpy.argsort(score) # [ 55  45  265  261 … 11  255  159]
#invert the indices, first the instances with the highest score
index = index[::-1] # [ 159  255  11 … 261  265  45  55 ]
#sort the class membership according to the indices
sort_pos = pos[index] # [ 1  1  1  1  1  0  1  1 …]
#cumulated sum
cpos = numpy.cumsum(sort_pos) # [ 1  2  3  4  5  5  6  7 … 99]
#recall column
rappel = cpos/npos # [ 1/99  2/99  3/99  4/99  5/99  5/99  6/99  7/99 … 99/99]
#nb. of instances into the test set
n = y_test.shape[0] # 300, il y a 300 ind. dans l'éch. test
#target size
taille = numpy.arange(start=1,stop=301,step=1) # [1  2  3  4  5  …  300]
#target size in percentage
taille = taille / n # [ 1/300  2/300  3/300 … 300/300 ]
```

The "scores" computed by the model seem quite good. There are a majority of positive instances for the highest scores.

```
#graphical representation with matplotlib
import matplotlib.pyplot as plt

#title and axis labels
plt.title('Courbe de gain')
plt.xlabel('Taille de cible')
plt.ylabel('Rappel')

#limits in horizontal and vertical axes
plt.xlim(0,1)
plt.ylim(0,1)

#tricks to represent the diagonal
plt.scatter(taille,taille,marker='.',color='blue')

#gains curve
plt.scatter(taille,rappel,marker='.',color='red')

#show the chart
plt.show()
```

The x-coordinate of the chart shows the percentage of the cumulative number of sorted data records according to the decreasing score value. The y-coordinate shows the percentage of the number of records that actually contain the selected target field value for the appropriate amount of records on the x-coordinate (see Gains chart).



Searching for estimator parameters

GRID SEARCH

```
#support vector machine
from sklearn import svm

#by default: RBF kernel and C = 1.0
mvs = svm.SVC()

#fit the model to the training sample
modele2 = mvs.fit(X_app,y_app)

#prediction on the test set
y_pred2 = modele2.predict(X_test)

#confusion matrix
print(metrics.confusion_matrix(y_test,y_pred2))

#success rate on the test set
print(metrics.accuracy_score(y_test,y_pred2)) # 0.67
```

Issue: Many machine learning algorithms are dependent to parameters that are not always obvious to determine to obtain the best performance on our dataset. E.g. 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*)

The method is not better than the default classifier (systematically predict the majority class value "negative"). Confusion matrix:

```
[[201   0]
 [ 99   0]]
```

The (SVM) method is unsuitable or the settings are not appropriate?

Grid search for searching the best parameters

```
#import the class
from sklearn import model_selection

#combination of parameters to evaluate
parametres = [{'C':[0.1,1,10],'kernel':['rbf','linear']}]

#cross-validation for 3 x 2 = 6 combinations
#accuracy rate is the performance measurement used
#mvs is the object form the svm.SVC class (cf. previous page)
grid = model_selection.GridSearchCV(estimator=mvs,param_grid=parametres,scoring='accuracy')

# launch searching - the calculations can be long
grille = grid.fit(X_app,y_app)

#result for each combination
print(pandas.DataFrame.from_dict(grille.cv_results_).loc[:,["params","mean_test_score"]])

# the best combination of C and kernel for our dataset
print(grille.best_params_) # {'C' : 10, 'kernel' : 'linear'}

# the performance of the best combination (success rate measured in cross-validation)
print(grille.best_score_) # 0.7564

#prediction with this best model i.e. {'C' : 10, 'kernel' : 'linear'}
y_pred3 = grille.predict(X_test)

#success rate on the test set
print(metrics.accuracy_score(y_test,y_pred3)) # 0.7833, the performance is similar to the one of logistic regression
```

We indicate the parameters to vary, scikit-learn combines them and measures performance in cross-validation for each combination.

	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

Selecting the most relevant features in a model

FEATURE SELECTION

Attribute selection (1/2)

Initial features (predictive attributes): pregnant, diastolic, triceps, bodymass, pedigree, age, plasma, serum.

```
#import  the LogisticRegression class
from sklearn.linear_model import LogisticRegression

#instantiate an object
lr = LogisticRegression()

#function for feature selection.
from sklearn.feature_selection import RFE
selecteur = RFE(estimator=lr)

#launch the selection process
sol = selecteur.fit(X_app,y_app)

#number of selected attributes
print(sol.n_features_) # 4 → 4 = 8 / 2 variables sélectionnées

#list of selected features
print(sol.support_) # [True False False True True False True False ]

# order of deletion
print(sol.ranking_) # [1  2  4  1  1  3  1  5]
```

Goal: detecting the subset of relevant features in order to obtain a simpler model, for a better interpretation, a shorter training time, and an enhanced generalization performance (1).
Approach: The RFE (recursive feature elimination) approach selects the features by recursively considering smaller and smaller sets of features. For the linear model, it is based on the value of the coefficients (the lowest one in absolute value is removed). The process continues until we reach the desired number of features. The variables must scaled (standardized or normalized) if we want to compare the coefficients.

Selected attributes: pregnant, bodymass, pedigree, plasma.

Serum was removed first, then **triceps**, then **age**, then **diastolic**. The remaining variables are indexed **1**.

```
# matrix for the selected attributes - training set
# we use the boolean vector sol.support_
X_new_app = X_app[:,sol.support_]
print(X_new_app.shape) # (468, 4) → 4 variables restantes

# fit the model on the selected attributes
modele_sel = lr.fit(X_new_app,y_app)

# matrix for the selected attributes – test set
X_new_test = X_test[:,sol.support_]
print(X_new_test.shape) # (300, 4)

# prediction on the test set
y_pred_sel = modele_sel.predict(X_new_test)

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

The resulting classifier is as good as (almost, 0793) the original model, but with half the number of attributes.

Course materials (in French)

http://eric.univ-lyon2.fr/~ricco/cours/cours_programmation_python.html

Python website

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

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

Scikit-learn

Machine Learning in Python

POLLS (KDnuggets)

Data Mining / Analytics Tools Used

Python, 4th in 2015

Primary programming language for Analytics, Data Mining, Data Science tasks

Python, 2nd in 2015 (next R)