Pratique: Matrice de Confusion

In [1]:
```
# Lire les données dans un dataframe
import pandas as pd
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-diabetes/pima-indians-diabetes.data
col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']
pima = pd.read_csv(url, header=None, names=col_names)
```

In [14]:
```
# print les 5 premieres lignes
pima.head()
```

Out[14]:

	pregnant	glucose	bp	skin	insulin	bmi	pedigree	age	label
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [ ]:
```
#shape
pima.shape
```

prediction du statut diabetique selon les prédicteurs

In [3]:
```
# definir les predicteurs X
feature_cols = ['pregnant', 'insulin', 'bmi', 'age']
X = pima[feature_cols]
#definir la cible y
y = pima.label
```

In [8]:
```
# split X et y ev training et testing
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

In [9]:
```
# train un modele logistic regression sur training
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
```

Out[9]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [10]:
```
# Faire des predictions avec testing
y_pred_class = logreg.predict(X_test)
```

In [11]:
```
# calcul du metrique accuracy
from sklearn import metrics
print("Accuracy:{} ".format(metrics.accuracy_score(y_test, y_pred_class)))
```

```
Accuracy:0.6927083333333334
```

In [12]:
```
# Afficher la distribution de classe du testing set
y_test.value_counts()
```

Out[12]:
```
0    130
1     62
Name: label, dtype: int64
```

In [15]:
```
# calcul des pourcentage de ones (1)
y_test.mean()
```

Out[15]:
```
0.6770833333333333
```

In [16]:
```
# calcul des pourcentage de ones (0)
1 - y_test.mean()
```

Out[16]:
```
0.6770833333333333
```

In [17]:
```
# Afficher les 20 premieres valeurs predites de statut

print('True:', y_test.values[0:20])
print('Pred:', y_pred_class[0:20])
```

```
True: [1 0 0 1 0 0 1 1 0 0 1 1 0 0 0 0 1 0 0 0]
Pred: [0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0]
```

Matrice de confusion

In [18]:
```
# Matrice de confusion: first argument is true values, second argument is predicted values
print(metrics.confusion_matrix(y_test, y_pred_class))
```

```
[[118  12]
 [ 47  15]]
```

- Definir d'abord l'hypothese nulle et l'hypothese alternative
- Terminologie pour lire la matrice de confusion
- True Positives (TP): On a prédit correctement qu'il a le diabete
- True Negatives (TN): On a prédit correctement qu'il n'a pas le diabete
- False Positives (FP): On a incorrectement predit qu'il a le diabete ("Type I error")
- False Negatives (FN): On a incorrectement predit qu'il n'a pas le diabetes ("Type II error")

In [21]:
```
# Obtenir les 4 valeurs de la matrice
confusion = metrics.confusion_matrix(y_test, y_pred_class)
TP = confusion[1, 1]
TN = confusion[0, 0]
FP = confusion[0, 1]
FN = confusion[1, 0]
```

In [22]:
```
# Classification Accuracy:
# On doit repondre à la question: Globalement, quel est le pourcentage où le modele est correct?
print((TP + TN) / float(TP + TN + FP + FN))
print(metrics.accuracy_score(y_test, y_pred_class))
```

```
0.692708333333
0.692708333333
```

In [23]:
```
#Classification Error:
# On doit repondre à la question: Globalement, quel est le pourcentage où le modele est incorrect?
print((FP + FN) / float(TP + TN + FP + FN))
print(1 - metrics.accuracy_score(y_test, y_pred_class))
```

```
0.307291666667
0.307291666667
```

In [ ]:
```
#Sensitivity:
# On doit repondre à la question: Quand la valeur vraie est positive, combien de fois la prediction est correcte?
# Ou How "sensitive" is the classifier to detecting positive instances?
#Aussi connu sous: "True Positive Rate" or "Recall"
```

In [24]:
```
print(TP / float(TP + FN))
print(metrics.recall_score(y_test, y_pred_class))
```

```
0.241935483871
0.241935483871
```

In [ ]:
```
#Specificity:
# On doit repondre à la question: Quand la valeur vraie est negative, combien de fois la prediction est correcte?
#OU How "specific" (or "selective") is the classifier in predicting positive instances?
```

In [25]:
```
print(TN / float(TN + FP))
```

```
0.907692307692
```

In [26]:
```
#False Positive Rate:
# On doit repondre à la question: Quand la valeur vraie est negative, combien de fois la prediction est incorrect
print(FP / float(TN + FP))
```

```
0.0923076923077
```

In [ ]:
```
# Precision:
# On doit repondre à la question: Quand une valeur positive est predite, combien de fois la prediction est correc
#OU How "precise" is the classifier when predicting positive instances?
```

In [27]:
```
print(TP / float(TP + FP))
print(metrics.precision_score(y_test, y_pred_class))
```

```
0.555555555556
0.555555555556
```

In [ ]:
```
de
Which metrics should you focus on?

Choice of metric depends on your business objective
Spam filter (positive class is "spam"): Optimize for precision or specificity because false negatives (spam goes
Fraudulent transaction detector (positive class is "fraud"): Optimize for sensitivity because false positives (no
```