# EVALUATION DES MODÈLES SUPERVISÉS

# RÉGRESSION - RAPPELS

Dans le cadre d'une régression, Y prend des valeurs continues. On cherche à comparer les vrais $y_i$ de l'ensemble de test avec les $\hat{y}_i$ prédits :

| $y_i$ | $\hat{y}_i$ |
|---|---|
| 3.0 | 2.5 |
| -0.5 | 0.0 |
| 2.0 | 2.0 |
| 7.0 | 8.0 |

# Erreur Quadratique Moyenne (MSE)

**Mean Squared Error (MSE)** : moyenne des carré des différences entre les vraies valeurs et les valeurs prédites :

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Correspond à la moyenne des distances des points de test à la droite de régression.

**Plus cette valeur est petite, meilleur est le modèle.**

```
from sklearn.metrics import mean_squared_error
y_true  =  [3,  -0.5,  2,  7]
y_pred  =  [2.5,  0.0,  2,  8]
mean_squared_error(y_true,  y_pred)
0.375
```

# Erreur absolue moyenne (MAE)

**Mean Absolute Error** : moyenne des valeurs absolues des différences entre les vraies valeurs et les valeurs prédites :

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

**Plus cette valeur est petite, meilleur est le modèle.**

```
from sklearn.metrics import mean_absolute_error
y_true  =  [3,  -0.5, 2, 7]
y_pred  =  [2.5, 0.0, 2, 8]
mean_absolute_error(y_true, y_pred)
0.5
```

# CLASSIFICATION - RAPPELS

Dans le cadre d'une classification, Y prend des valeurs discrètes (dans cet exemple, 3 classes possibles). On cherche à comparer les vrais $y_i$ de l'ensemble de test avec les $\hat{y}_i$ prédits :

| $y_i$ | $\hat{y}_i$ |
|---|---|
| 0 | 0 |
| 1 | 2 |
| 2 | 2 |
| 1 | 0 |

# TAUX DE BONNE CLASSIFICATION

**Accuracy** : pourcentage des observations pour lesquelles le modèle a bien prédit.

$$acc = \frac{1}{n}\sum_{i=1}^{n} \delta(\hat{y}_i = y_i)$$

**Plus cette valeur est grande (entre 0 et 1), meilleur est le modèle.**

```
from sklearn.metrics import accuracy_score
y_pred  =  [0, 1, 2, 1]
y_true  =  [0, 2, 2, 0]
accuracy_score(y_true,  y_pred)
0.5
```

# MATRICE DE CONFUSION - CAS BINAIRE

Permet en un coup d'oeil de **diagnostiquer** les erreurs faites par l'algorithme.

| | | **Prédit** | |
|---|---|---|---|
| | | 0 | 1 |
| **Vrai** | 0 | TN (Vrais négatifs) | FP (faux positifs) |
| | 1 | FN (faux négatifs) | TP (Vrais positifs) |

```
from sklearn.metrics import confusion_matrix
y_true  =  [1, 0, 1, 0, 0, 1]
y_pred  =  [0, 0, 1, 1, 1, 1]
confusion_matrix(y_true,  y_pred)
array([[1, 1],
       [2, 2])
```

# MATRICE DE CONFUSION MULTICLASSES

| | | Prédit | | |
|---|---|---|---|---|
| | | 1 | 2 | 3 |
| Vrai | 1 | Vrais 1 | 1 prédits 2 | 1 prédits 3 |
| | 2 | 2 prédits 1 | Vrais 2 | 2 prédits 3 |
| | 3 | 3 prédits 1 | 3 prédits 2 | Vrais 3 |

```
from sklearn.metrics import confusion_matrix
y_true  =  [1, 2, 2, 3, 2, 3]
y_pred  =  [1, 3, 2, 1, 2, 1]
confusion_matrix(y_true, y_pred)
array([[1, 0, 0],
       [0, 2, 1],
       [2, 0, 0])
```

# TAUX DE VRAIS POSITIFS

**Seulement valable dans le cas binaire.**

**True Positive Rate (TPR)** : pourcentage des observations positives prédites positives.

$$TPR = \frac{\overbrace{TP}^{\text{Positifs prédits positifs}}}{\underbrace{TP + FN}_{\text{Tous les positifs de l'ensemble}}}$$

```
import numpy as np
y_pred  =  np.array([0,  1,  1,  0])
y_true  =  np.array([0,  1,  0,  0])
tpr = (y_true + y_pred == 2).sum()*1.0 / (y_true == 1).sum()
tpr
>> 1.0
```

# TAUX DE VRAIS NÉGATIFS

**Seulement valable dans le cas binaire.**

**True Negative Rate (TNR)** : pourcentage des observations négatives prédites négatives.

$$TNR = \frac{\overbrace{TN}^{\text{Négatifs prédits négatifs}}}{\underbrace{TN + FP}_{\text{Tous les négatifs de l'ensemble}}}$$

```
import numpy as np
y_pred  =  np.array([0,  1,  1,  0])
y_true  =  np.array([0,  1,  0,  0])
tnr = (y_true + y_pred == 0).sum()*1.0 / (y_true == 0).sum()
tnr
>> 0.6666667
```

# TAUX DE BONNE CLASSIFICATION ÉQUILIBRÉ

**Seulement valable dans le cas binaire.**

**Balanced Accuracy** : moyenne du TPR et du TNR. Utile en cas de classes très déséquilibrées.

$$bal_{acc} = \frac{TPR + TNR}{2}$$

```
import numpy as np
y_pred  =  np.array([0,  1,  1,  0])
y_true  =  np.array([0,  1,  0,  0])
tpr = (y_true + y_pred == 2).sum()*1.0 / (y_true == 1).sum()
tnr = (y_true + y_pred == 0).sum()*1.0 / (y_true == 0).sum()
bal_acc = (tpr + tnr) / 2
bal_acc
>> 0.8333333333333326
```

# PRÉCISION ET RECALL

**Precision** : taux de réels positifs parmi les positifs prédits.

$$precision = \frac{TP}{TP + FP}$$

**Recall** : taux de positifs prédits parmi les réels positifs = TPR

$$recall = \frac{TP}{TP + FN}$$

```
from sklearn.metrics import precision_score, recall_score
y_pred  =  np.array([0,  1,  1,  0])
y_true  =  np.array([0,  1,  0,  0])
precision_score(y_true, y_pred)
>>>  0.5
recall_score(y_true,  y_pred)
>>>  1.0
```

# $F_1$-SCORE

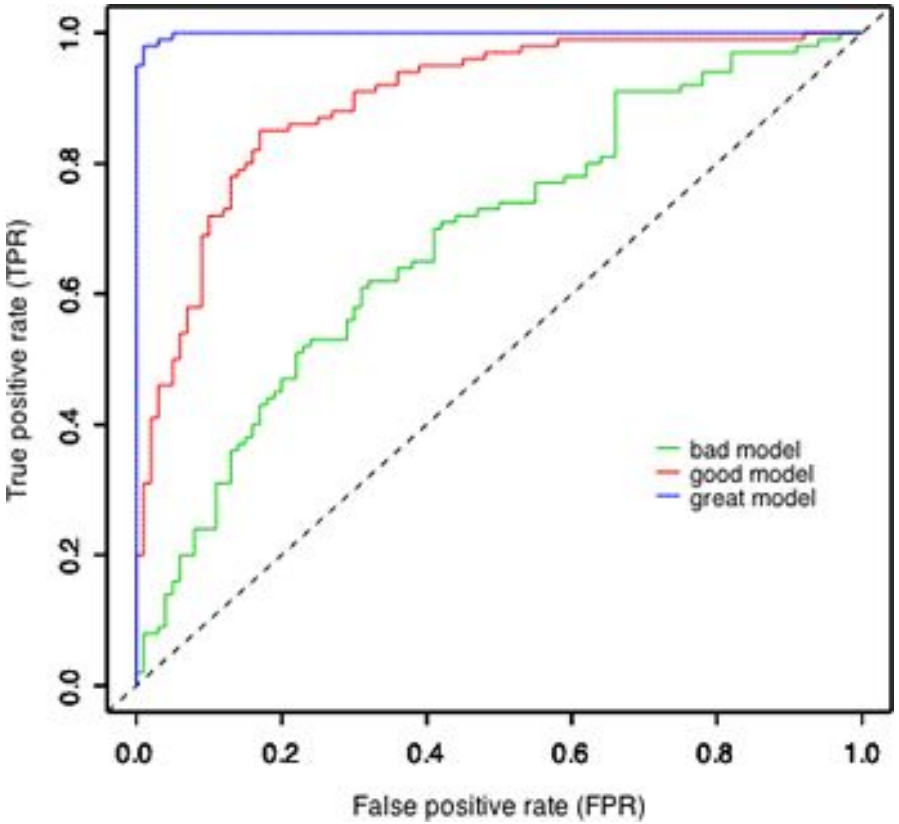
**$F_1$-score** : moyenne harmonique du recall et de la précision.

$$f_1 = 2 \times \frac{precision \times recall}{precision + recall}$$

**Plus cette valeur est grande (entre 0 et 1), meilleur est le modèle.**

```
from sklearn.metrics import f1_score
y_pred  =  np.array([0,  1,  1,  0])
y_true  =  np.array([0,  1,  0,  0])
f1_score(y_true,  y_pred)
>>>  0.66666666666
```

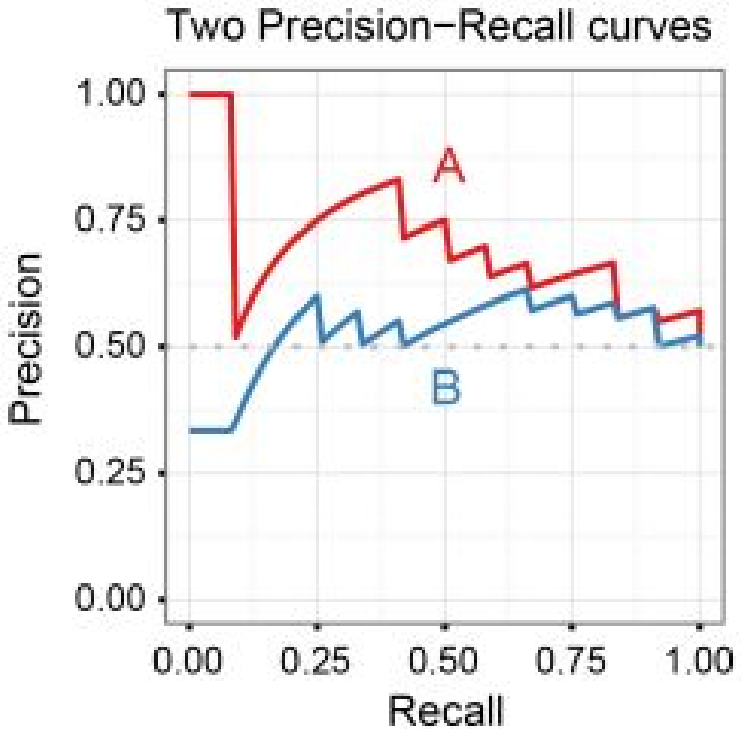
# COURBE ROC




*Source : jxieeducation.com*

**ROC** (Receiver Operating Characteristics) : progrès de l'algorithme lorsqu'on fait varier le seuil de discrimination (valeur à partir de laquelle ŷ vaut 1). Cette courbe est forcément croissante.

# COURBE PRÉCISION-RECALL




Progrès de l'algorithme lorsqu'on fait varier le seuil de discrimination (valeur à partir de laquelle ŷ vaut 1). La meilleure courbe est celle qui permet d'obtenir le meilleur compromis précision/recall.

*Source : classval.wordpress.com*