# Apprentissage supervisé, deuxième partie

# Rappels

Cours précédent :

- Concept de l'**apprentissage supervisé** : montrer à l'ordinateur des exemples pour qu'il apprenne une règle.
- **Phase d'apprentissage** : on connaît l'entrée X et la sortie/réponse Y, on cherche la fonction qui les lie $Y = f(X)$
- **Optimisation des paramètres** : pendant la phase d'apprentissage, grâce à la validation croisée
- **Phase de test** : on fait prédire des valeurs à l'algorithme et on les compare aux vraies valeurs pour estimer sa performance.
- **Deux types de modèles** : algorithmiques (semaine dernièreÂ° et probabilistes (cette semaine).

# Outline


1. Modèles probabilistes
   - Naive Bayes
   - Régression linéaire
   - Régression pénalisée
   - Régression logistique

2. Evaluation des modèles supervisés
   - Modèles de régression
   - Modèles de classification

# Outline


1. Modèles probabilistes
   - Naive Bayes
   - Régression linéaire
   - Régression pénalisée
   - Régression logistique
2. Evaluation des modèles supervisés
   - Modèles de régression
   - Modèles de classification

# Probabilités conditionnelles

Si A et B sont deux événements, la probabilité que A se produise conditionnellement au fait que B se produise se dit **probabilité de A sachant B** s'écrit : $P(A|B)$.

Exemple 1:

- A : "il pleut aujourd'hui"
- B : "il pleuvait hier"
- $P(A|B)$ : probabilité qu'il pleuve aujourd'hui *sachant* qu'il pleuvait hier.
- $P(B|A)$ : probabilité qu'il ait plu hier *sachant* qu'il pleut aujourd'hui.
- $P(A, B) = P(A \cap B)$ : **probabilité jointe** qu'il ait plu hier **et** qu'il pleuve aujourd'hui.

# Probabilités conditionnelles

Si A et B sont deux événements, la probabilité que A se produise conditionnellement au fait que B se produise se dit **probabilité de A sachant B** s'écrit : $P(A|B)$.

Exemple 2:

- A : "j'achète"
- B : "j'ai vu une pub"
- $P(A|B)$ : probabilité que j'achète *sachant* que j'ai vu une pub .
- $P(B|A)$ : probabilité que j'aie vu une pub *sachant* que j'ai acheté.
- $P(A, B) = P(A \cap B)$ : **probabilité jointe** que j'aie vu une pub **et** que j'aie acheté.

# Calcul des probabilités conditionnelles

Si A et B sont deux événements, alors:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

| Jour | Lundi | Mardi | Mercredi | Jeudi | Vendredi |
|---|---|---|---|---|---|
| Hier | Pluie | Pluie | Soleil | Soleil | Pluie |
| Aujourd'hui | Pluie | Soleil | Soleil | Pluie | Soleil |

- **Probabilités simples** : $P(\text{Pluie aujourd'hui}) = 2/5$, $P(\text{Pluie hier}) = 3/5$
- **Probabilité jointe** : $P(\text{Pluie aujourd'hui}, \text{Pluie hier}) = 1/5$
- **Probabilités conditionnelles** :

$$P(\text{Pluie auj}|\text{Pluie hier}) = \frac{P(\text{Pluie auj}, \text{Pluie hier})}{P(\text{Pluie hier})} = \frac{1/5}{3/5} = 1/3$$

$$P(\text{Pluie hier}|\text{Pluie auj}) = \frac{P(\text{Pluie auj}, \text{Pluie hier})}{P(\text{Pluie auj})} = \frac{1/5}{2/5} = 1/2$$

# Calcul des probabilités conditionnelles

Si A et B sont deux événements, alors:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

| Obs. | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Pub | Oui | Non | Oui | Oui | Oui | Non | Oui | Oui |
| Acheté | Non | Oui | Non | Oui | Non | Non | Non | Non |

- **Probabilités simples** : $P(\text{Pub}) = 6/8$, $P(\text{Acheté}) = 2/8$
- **Probabilité jointe** : $P(\text{Acheté}, \text{Pub}) = 1/8$
- **Probabilités conditionnelles** :

$$P(\text{Acheté}|\text{Pub}) = \frac{P(\text{Acheté}, \text{Pub})}{P(\text{Pub})} = \frac{1/8}{6/8} = 1/6$$

$$P(\text{Pub}|\text{Acheté}) = \frac{P(\text{Acheté}, \text{Pub})}{P(\text{Acheté})} = \frac{1/8}{2/8} = 1/2$$

# Le théorème / la règle de Bayes

Si A et B sont deux événements:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \left( = \frac{P(A \cap B)}{P(B)} \right)$$

car : $P(A \cap B) = P(A|B)P(B) = P(B|A)P(A)$

- Dans une université, il y a 34% de femmes.
- Parmi les étudiants en informatique, 22% sont des femmes.
- Les étudiants en informatique représentent 20% de l'université.
- Quellle est la proportion d'étudiantes en informatique parmi les femmes de l'université ?
- On cherche $P(info|femme)$

$$P(info|femme) = \frac{P(femme|info)P(info)}{P(femme)} = \frac{0,22 \times 0,20}{0,34} = 12,9\%$$

# Naive Bayes : phase d'apprentissage (ex du spam)

Pour chaque mot W rencontré dans l'ensemble d'apprentissage, on calcule la probabilité qu'un message soit un spam **sachant** qu'il contient le mot W (**spamicité** du mot W):

$$P(S|W) = \frac{\overbrace{P(W|S)}^{\%\text{ de spams contenant W}} \quad \overbrace{P(S)}^{\%\text{ de spams}}}{\underbrace{P(W)}_{\text{fréquence d'apparition du mot}}}$$

# Naive Bayes : phase de test (ex. du spam)

Pour un message de l'ensemble de test, on calcule la **probabilité que le message soit un spam** sachant qu'il contient les mots $W_1, W_2, W_3 \ldots W_n$ :

$$P(S|W_1, W_2 \ldots W_n) = \frac{\overbrace{p_1 p_2 \ldots p_n}^{\text{spamicités des mots du message}}}{\underbrace{p_1 p_2 \ldots p_n}_{\text{spamicité}} + \underbrace{(1-p_1)(1-p_2)\ldots(1-p_n)}_{\text{non spamicité}}}$$

avec $p_1 = P(S|W_1), p_2 = P(S|W_2), \ldots, p_n = P(S|W_n)$

# Remarques

- Le classifieur Naive Bayes considère que **tous les mots sont indépendants**. Autrement dit, il regarde l'effet des mots un par un, sans se soucier des autres. C'est la raison pour laquelle il est **naïf**.
- Et pourtant ! Existant depuis longtemps (1996) et malgré son caractère naïf, il reste très utilisé, en particulier pour la classification du spam et donne de **bons résultats**.
- Il est recommandé de **supprimer les mots apparaissant trop peu ou trop souvent** qui peuvent poser problème ou l'induire en erreur.
- Il peut classifier les documents dans plusieurs catégories, le cas binaire du spam étant un cas particulier.

# Conclusion sur Naive Bayes

**Avantages :**

- Modèle très simple.
- Rapide car très peu de calculs.
- Efficace dans certains cas.

**Inconvénients :**

- Sa naïveté ne s'applique pas à tous les problèmes.
- Il ne dispense pas (au contraire!) de préprocesser le texte (comme tous les algorithmes).

**En Python :**

```
from sklearn.naive_bayes import BernoulliNB, Multinomia
model = BernoulliNB()
model.fit(Xtrain,Ytrain)
predictions = model.predict(Xtest)
```

# Outline

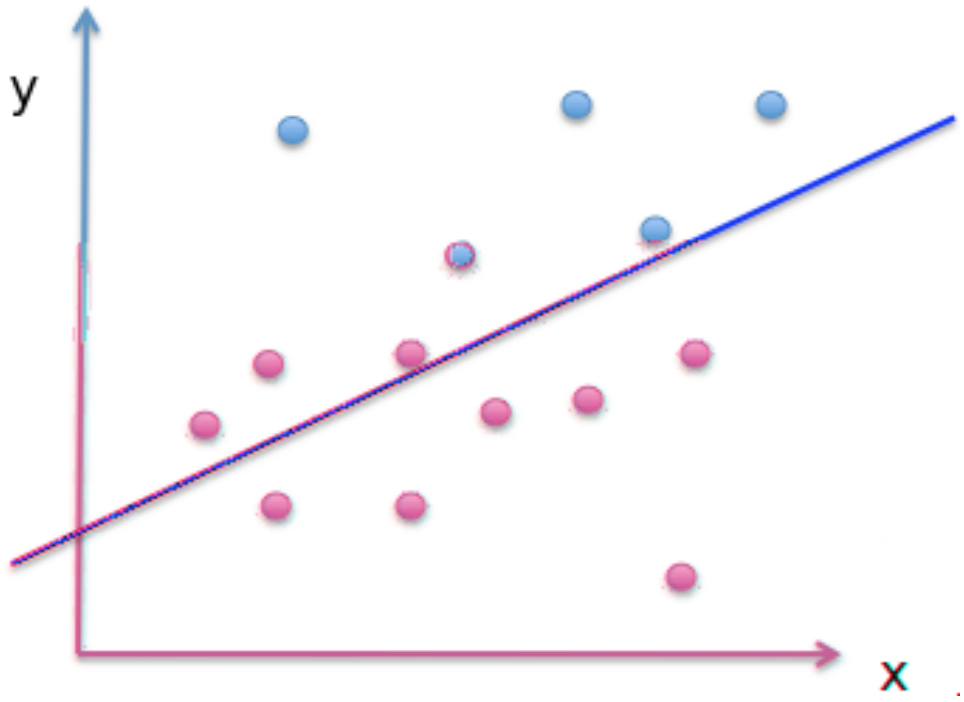

1. Modèles probabilistes
   - Naive Bayes
   - Régression linéaire
   - Régression pénalisée
   - Régression logistique
2. Evaluation des modèles supervisés
   - Modèles de régression
   - Modèles de classification

# Régression linéaire simple

- A partir de $n$ exemples, trouver la droite qui passe en leur "centre".
- $\beta_0$ : biais (valeur en $x = 0$)
- $\beta_1$ : pente

$$\text{réponse} = \beta_0 + \beta_1 \times (\text{variable dépendante}) + \text{bruit}$$

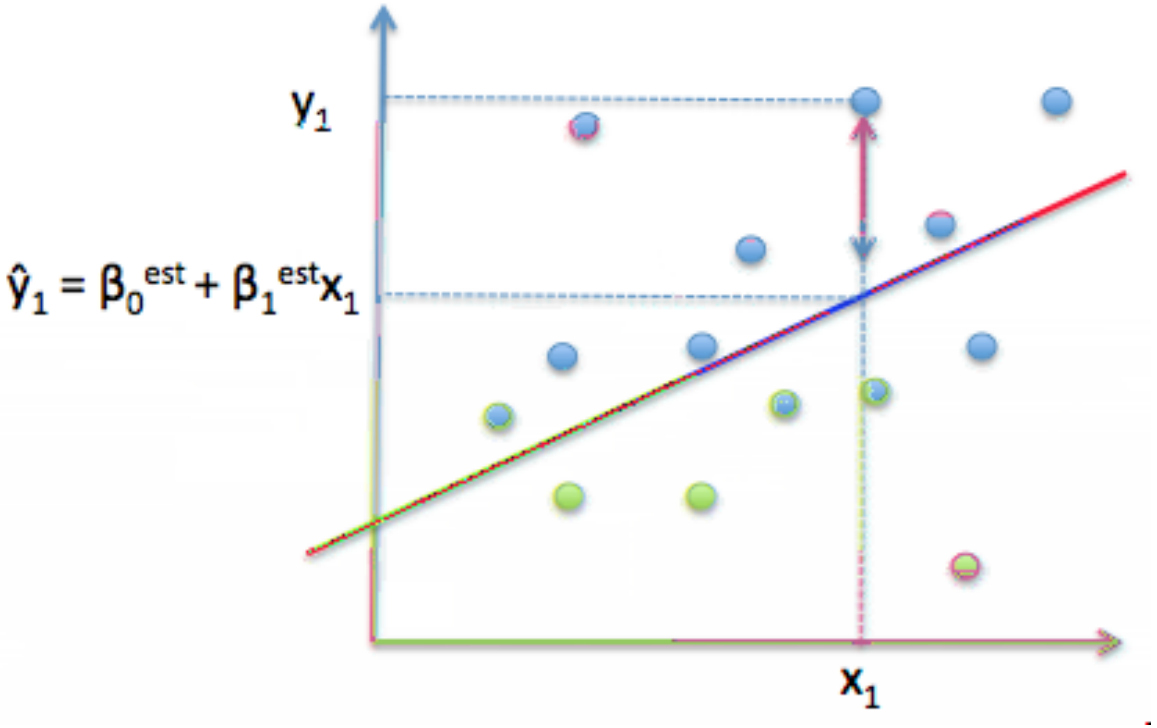
$$Y = \beta_0 + \beta_1 x + \varepsilon$$



# Estimation

Objectif : minimiser la **variance résiduelle** (i.e. $\varepsilon$) trouver la meilleure droite :

$$\min_f \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$
$$\Leftrightarrow \quad \min_f \sum_{i=1}^{n} (y_i - f(x_i))^2$$
$$\Leftrightarrow \quad \min_{\beta_0,\beta_1} \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i)^2$$



# Régression linéaire multiple

- En ajoutant de nouvelles variables :

$$\begin{aligned} \text{réponse} &= \beta_0 + \beta_1 \times (\text{variable 1}) + \beta_2 \times (\text{variable 2}) + \cdots + \text{bruit} \\ Y &= \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \varepsilon \end{aligned}$$

- Forme générale avec $p$ variables :

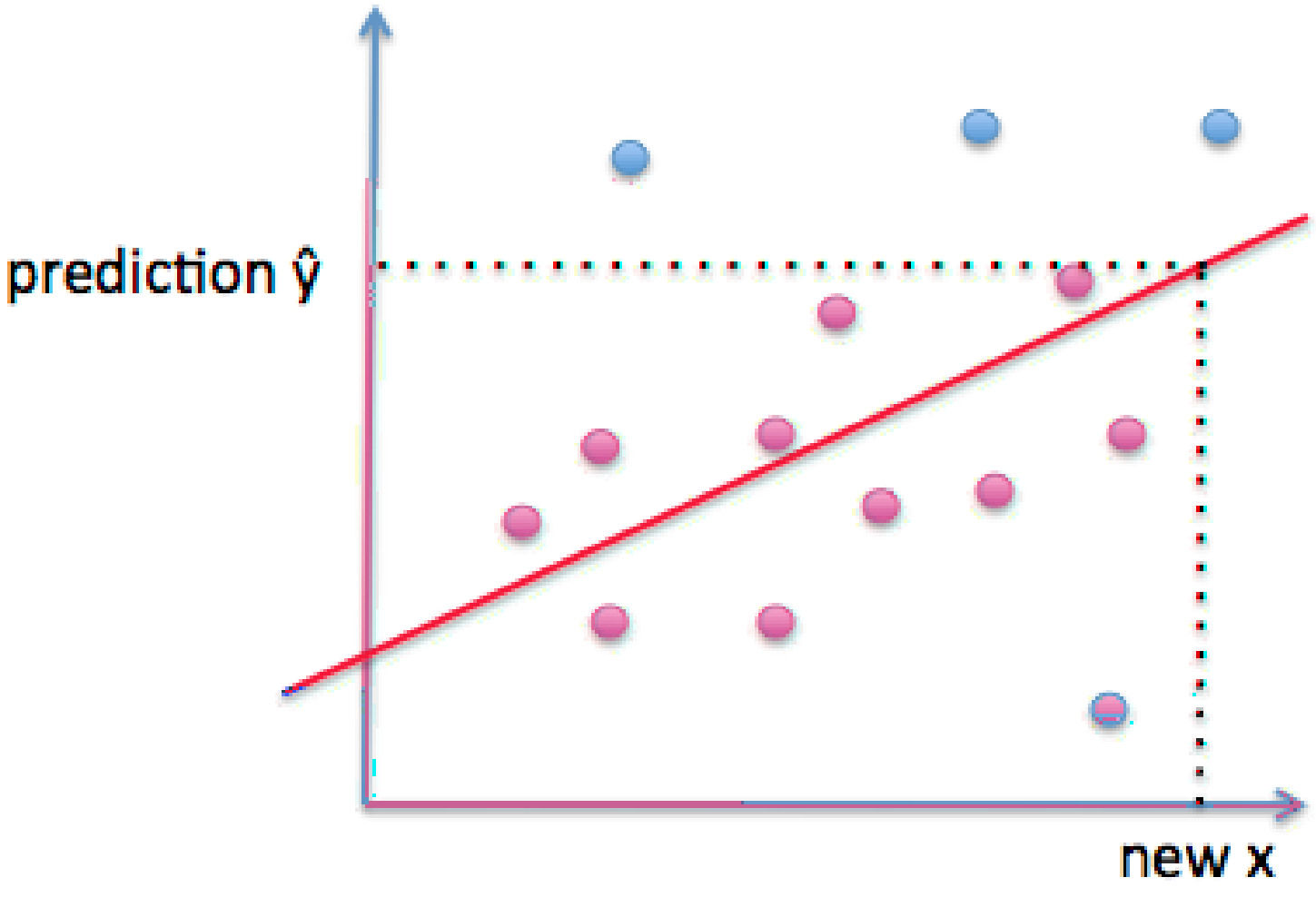
$$Y = \beta_0 + \sum_{j=1}^{p} \beta_j x_j + \varepsilon = \beta_0 + X\beta + \varepsilon$$

- Solution :

$$(\hat{\beta}_0, \hat{\beta}) = \underbrace{\arg \min_{\beta_0 \in \mathbb{R}, \beta \in \mathbb{R}^p}}_{\text{Trouver } \beta_0 \text{ et le vecteur } \beta \text{ qui minimisent...}} \underbrace{\sum_{i=1}^{n} (y_i - \beta_0 - X_i\beta)^2}_{\text{... la fonction de perte}}$$

# Sur l'ensemble de test

Quand un nouveau point *x* est donné, on peut **prédire** le $\hat{y}$ correspondant:



# Interprétation

**Supposons un modèle :**

$$\text{salaire} = \beta_0 + \beta_1 \times \text{expérience} + \beta_2 \times \text{études} + \text{bruit}$$

**Et supposons que les résultats soient :**

# L'interprétation est-elle valable ?

- On cherche à expliquer la **variance totale** (total sum of squares : TSS): $TSS = \sum_{i=1}^{n}(y_i - \overline{y})^2$
- On calcule la **variance expliquée** (explained sum of squares : ESS): $ESS = \sum_{i=1}^{n}(\hat{y}_i - \overline{y})^2$
- On veut minimiser la **variance résiduelle** (residual sum of squares : RSS): $RSS = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$

Et on a : $TSS = RSS + ESS$

Le **pourcentage de variance expliquée** donne donc une indication sur la qualité de la régression qu'on appelle $R^2$ (à calculer également sur l'ensemble de test) :

$$R^2 = \frac{ESS}{TSS} = 1 - \frac{RSS}{TSS}$$

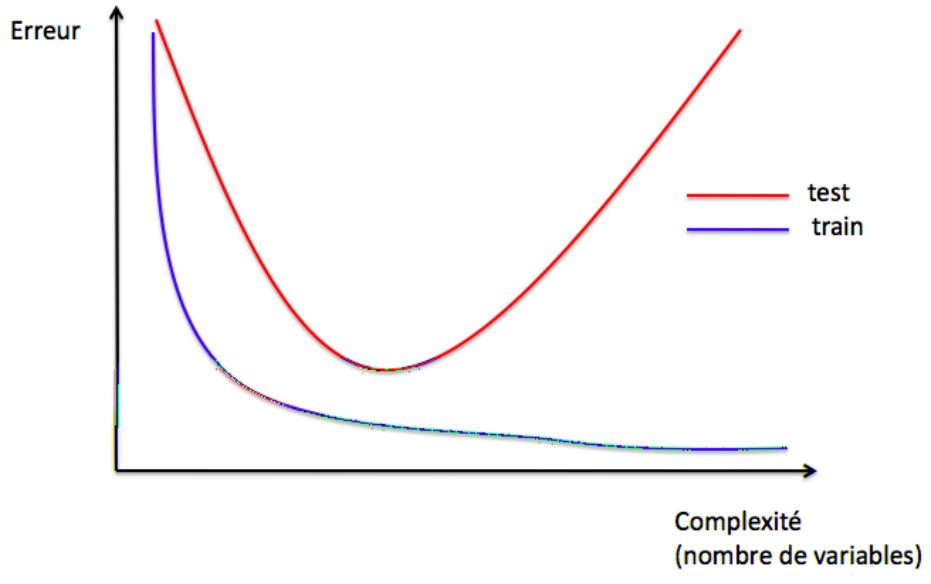
Le $R^2$ se trouve entre 0 et 1 :

- 0 : rien n'est expliqué par les variables
- 1 : tout est expliqué par les variables

# Limites

Le modèle linéaire est séduisant par sa simplicité mais vite limité :

- La solution demande l'inversion de la matrice $X^T X$ qui n'est pas inversible si l'on a **plus de variables que d'observations**.
- Si lde**s** variables sont **corrélées**, cela va perturber le modèle : deux variables très ressemblantes n'auront pas forcément les mêmes poids.
- Plus on ajoute de variables, plus le modèle devient **instable** et on risque donc le **sur-apprentissage**.



# Outline


1. Modèles probabilistes
   - Naive Bayes
   - Régression linéaire
   - Régression pénalisée
   - Régression logistique
2. Evaluation des modèles supervisés
   - Modèles de régression
   - Modèles de classification

# Pénalisation

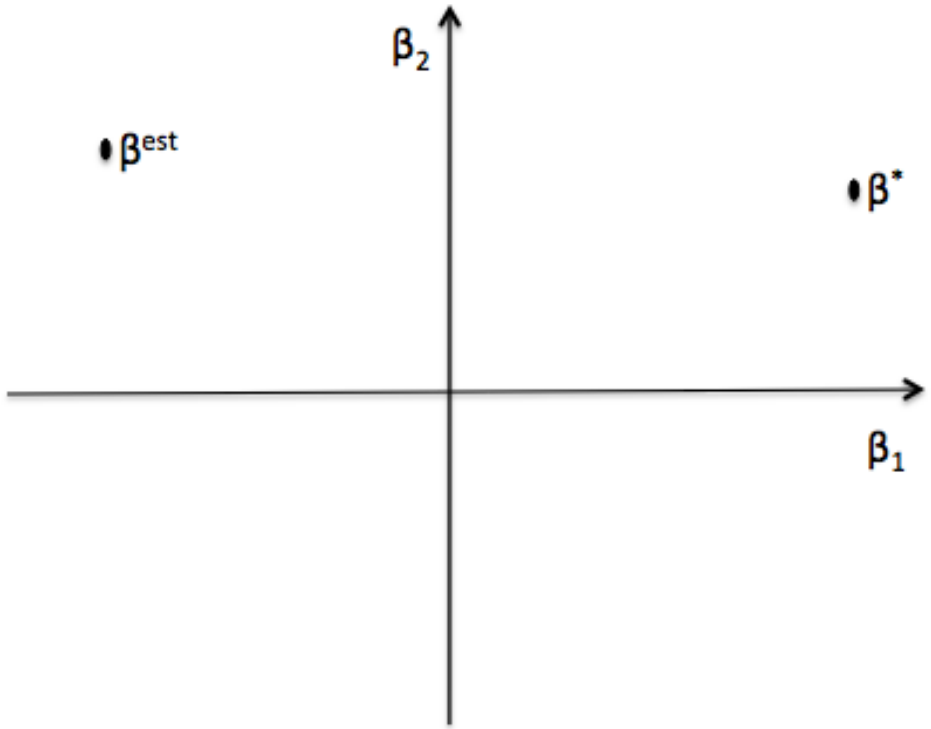
- Un model trop compliqué peut être parfait sur l'ensemble d'apprentissage.
- Mais il risque d'être **mauvais** sur de nouvelles données.
- Une solution : **pénaliser** la complexité du modèlel.
- Le problème devient :

$$\hat{f} = \arg\min_{\beta} \sum_{i=1}^{n} loss(y_i, f(x_i))$$
$$\text{s.t. f satisfasse des contraintes : } \Omega(f) \leq T$$

# Régression Ridge

- On suppose $f$ **linéaire**, i.e. $y = f(x) = \beta_0 + X\beta$.
- A résoudre :

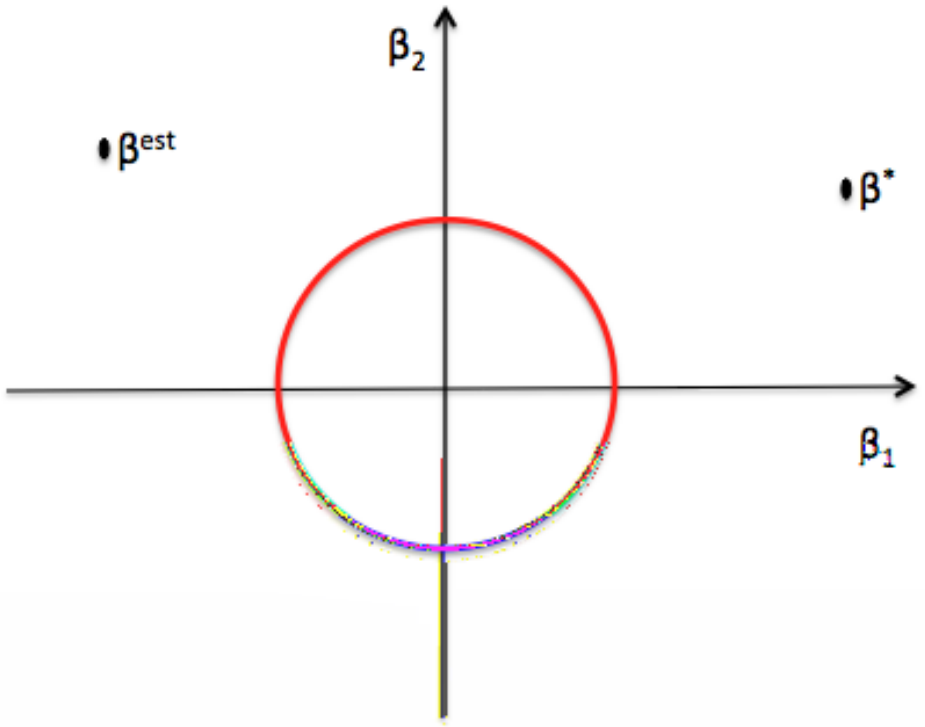
$$\hat{\beta} = \arg\min_{\beta} \sum_{i=1}^{n} (y_i - \beta_0 - x_i\beta)^2$$



# Régression Ridge

- On ajoute une **contrainte Ridge** : l'amplitude des poids doit être faible.
- A résoudre :

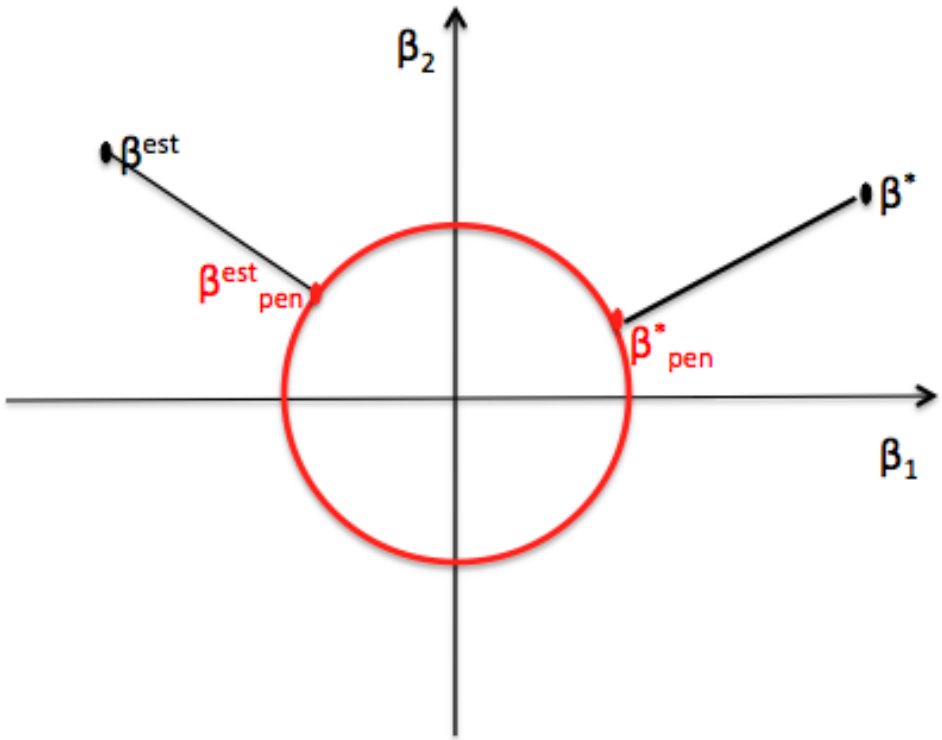
$$\hat{\beta} = \arg\min_{\beta} \sum_{i=1}^{n} (y_i - \beta_0 - x_i\beta)^2$$
$$\text{s.t. } \sum_{j=1}^{p} \beta_j^2 \leq T$$



# Régression Ridge

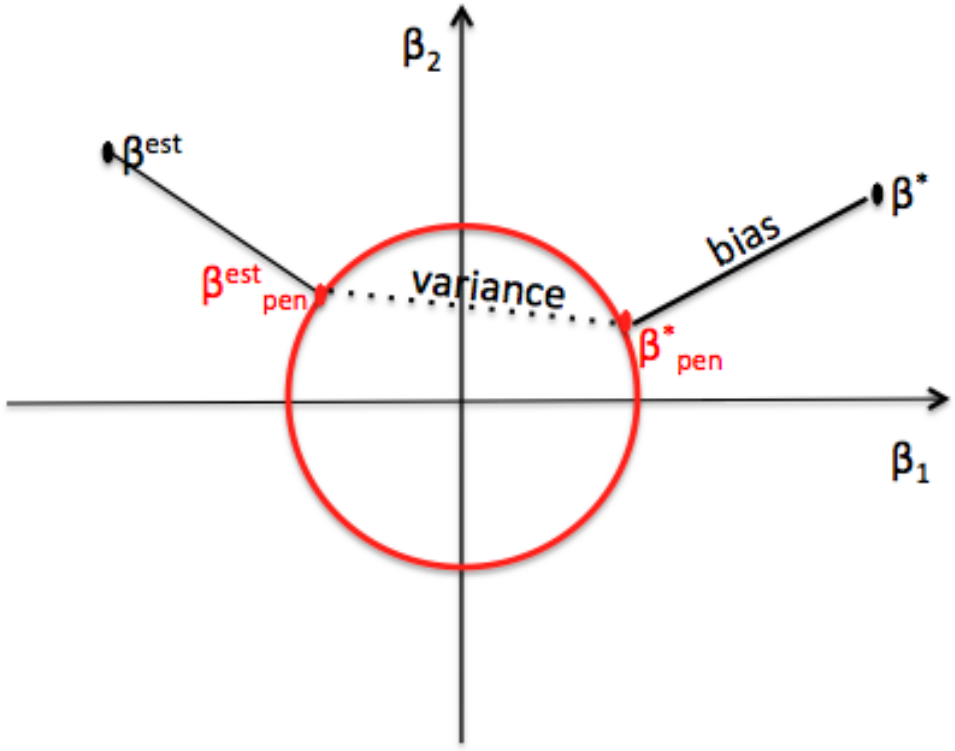
- Les poids $\beta$ sont maintenant **forcés de satisfaire cette contrainte**.
- A résoudre :

$$\hat{\beta} = \arg\min_{\beta} \sum_{i=1}^{n} (y_i - \beta_0 - x_i\beta)^2$$
$$\text{s.t. } \sum_{j=1}^{p} \beta_j^2 \leq T$$



# Régression Ridge

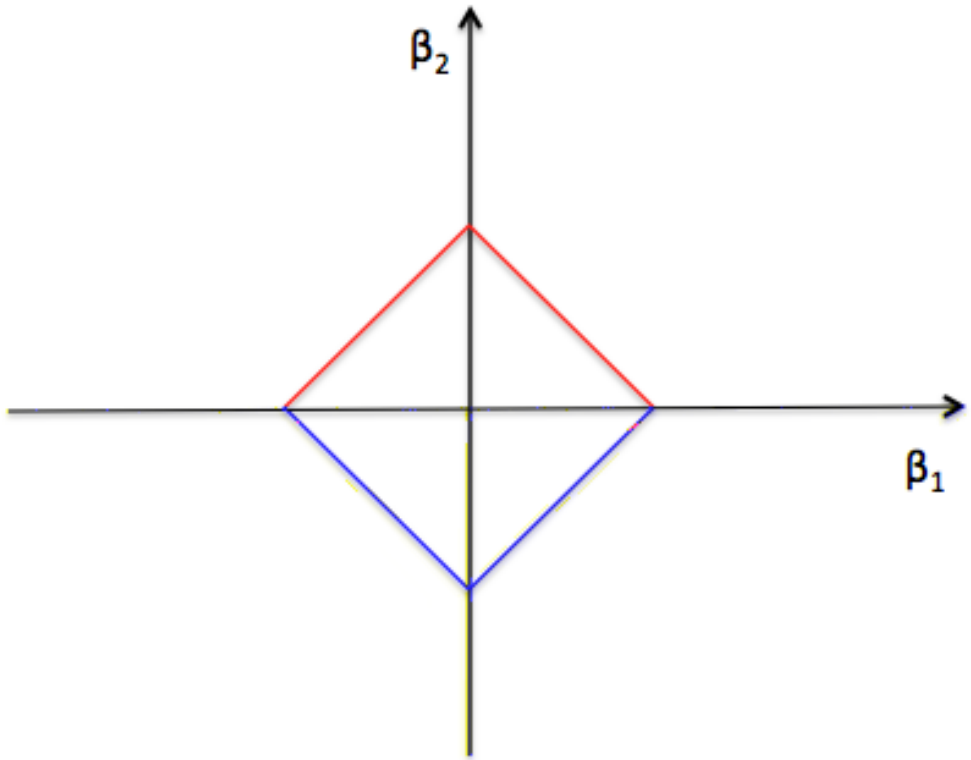
- **Biais** : à quel point le modèle est faux
- **Variance** : à quel point l'estimation du modèle est fausse.
- Pénaliser la complexité permet d'obtenir un bon **compromis biais/variance**.



# Régression Lasso

- **Contrainte Lasso** : on restreint à la fois l'amplitude des poids et le nombre des variables.
- A résoudre :

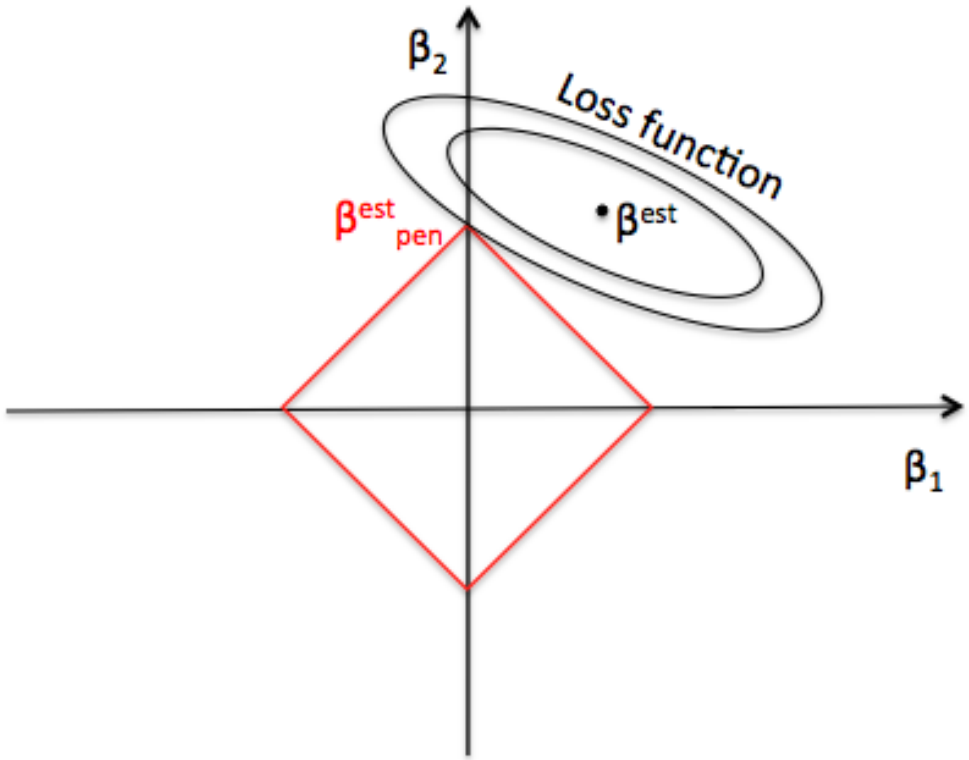
$$\hat{\beta} = \arg\min_{\beta} \sum_{i=1}^{n} (y_i - \beta_0 - x_i\beta)^2$$
$$\text{s.t. } \sum_{j=1}^{p} |\beta_j| \leq T$$



# Régression Lasso

- De nombreuses solutions vont tomber sur une **singularité** and seront donc **parcimonieuses** (sparse).
- A résoudre :

$$\hat{\beta} = \arg\min_{\beta} \sum_{i=1}^{n} (y_i - \beta_0 - x_i\beta)^2$$
$$\text{s.t. } \sum_{j=1}^{p} |\beta_j| \leq T$$



# Elastic Net

- **Ridge**:
  - $||X_i\beta_i - X_j\beta_j||_2 \leq ||\beta||_2||X_i - X_j||_2$: deux exemples similaires auront deux prédictions similaires.
  - mais: **pas sparse** (toutes les variables sont utilisées).
- **Lasso**:
  - solutions **sparse**
  - mais: très instable quand les variables sont correlées.
- **Alternative: Elastic Net**
  - problème:

$$\hat{\beta} = \arg\min \sum_{i=1}^{n}(y_i - \beta_0 - x_i\beta)^2$$
$$\text{s.t. } \sum_{j=1}^{p}|\beta_j| \leq T_1$$
$$\sum_{j=1}^{p}\beta_j^2 \leq T_2$$

  - combine les avantages de Ridge et Lasso
  - sparse tout en gardant un effet de "groupe"

# D’autres pénalisations possibles

- **Group Lasso** : force le modèle à sélectionner des groupes de variables.
- **Graph Lasso** : sélectionne des groupes de variables qui sont proches sur un graphe.
- **K-support norm** : sélectionne des groupes non prédéfinis.
- On appelle ces méthodes des méthodes de **pénalisation structurée**.
- Quand on a une idée de la manière dont fonctionne les variables, on peut crér sa propre pénalité.

# Conclusions sur la régression linéaire

**Avantages :**

- Modèle simple et facile à estimer.
- Interprétable.
- On peut le pénaliser de manière spécifique pour chaque problème.

**Inconvénients :**

- Régression linéaire non pénalisée : impossible en grande dimension.
- La pénalisation contraint le modèle à une petite partie de l'espace : il faut être sûr de soi !

**En Python :**

```
from sklearn.linear_model import LinearRegression,
Lasso, Ridge, ElasticNet
reg = LinearRegression()
reg.fit(Xtrain,Ytrain)
lasso = Lasso(alpha = 1)
lasso.fit(Xtrain,Ytrain)
```

# Outline

1. Modèles probabilistes
   - Naive Bayes
   - Régression linéaire

# Régression logistique binaire

- Dans la régression linéaire, on prédit une variable $Y$ continue
- La régression logistique permet en fait de faire de la **classification**.
- On considère ici la classification binaire : $Y \in \{0, 1\}$.

# Principe

Comme d'habitude, $X$ est l'input, $Y$ l'output. La régression logistique binaire repose sur le modèle suivant :

$$P(Y = 1|X) = \frac{1}{1 + \exp^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 ...)}}$$

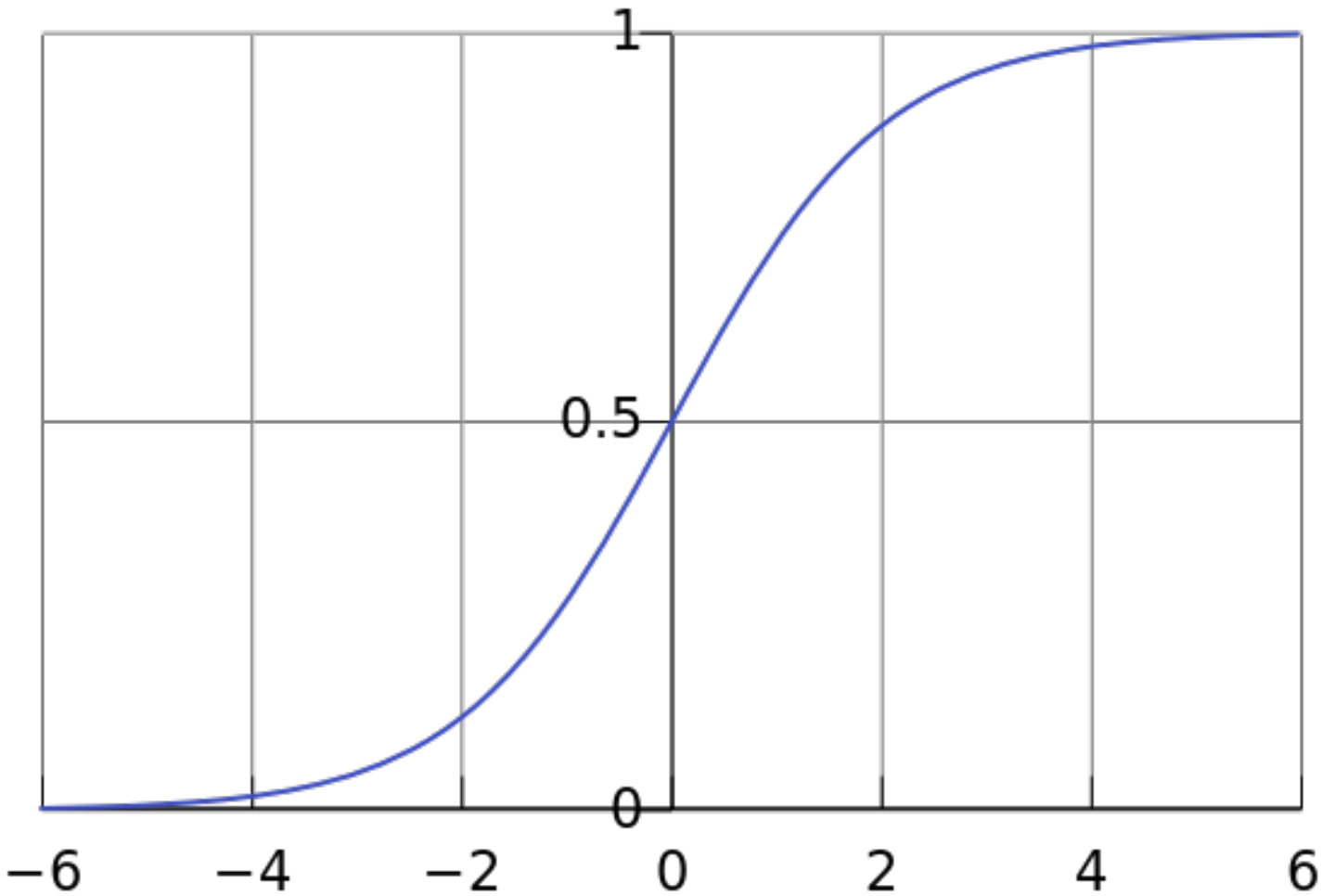
On réfléchit en termes de **rapport de probabilités** ou encore de **odd ratios** :

$$\frac{P(Y = 1|X)}{P(Y = 0|X)} = \exp^{\beta_0 + \beta_1 x_1 + \beta_2 x_2 ...}$$

Ou encore :

$$\ln\left(\frac{P(Y = 1|X)}{P(Y = 0|X)}\right) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 ...$$

# La fonction logit




*Source : wikipedia.com*

**La régression logistique revient à appliquer cette fonction à une régression linéaire simple pour "pousser" les valeurs vers 0 et 1.**

# Conclusions sur la régression logistique

## Avantages

- Une vision probabiliste de la classification
- Interprétation de chaque variable de manière probabiliste
- Simple et rapide à estimer

## Inconvénients

- Mêmes que la régression linéaire
- En particulier, mauvais résultats avec beaucoup de variables
- Mais il est possible de la pénaliser.

## En Python :

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(Xtrain, Ytrain)
model.predict(Xtest)
```

# Outline

1. Modèles probabilistes
   - Naive Bayes
   - Régression linéaire
   - Régression pénalisée
   - Régression logistique

2. Evaluation des modèles supervisés
   - Modèles de régression
   - Modèles de classification

# Outline


1. Modèles probabilistes
   - Naive Bayes
   - Régression linéaire
   - Régression pénalisée
   - Régression logistique
2. Evaluation des modèles supervisés
   - Modèles de régression
   - Modèles de classification

# Rappel

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 :

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

**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
```

# Outline


1. Modèles probabilistes
   - Naive Bayes
   - Régression linéaire
   - Régression pénalisée
   - Régression logistique
2. Evaluation des modèles supervisés
   - Modèles de régression
   - Modèles de classification

# Rappel

Dans le cadre d'une classification, $Y$ prend des valeurs discrètes. 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 |
| 2 | 1 |
| 1 | 2 |
| 3 | 3 |

# Pourcentage de bonne classification

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

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

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

```
from sklearn.metrics import accuracy_score
y_pred = [0, 2, 1, 3]
y_true = [0, 1, 2, 3]
accuracy_score(y_true, y_pred)
0.5
```

# Matrice de confusion

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) |

En python :

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

# Taux de vrais positifs

**Seulement valable dans le cas binaire.**

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

$$TPR = TP/(TP + FN)$$

```
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

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

$$TNR = TN/(TN + FP)$$

```
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
>>> 0.6666666667
```

# Taux de bonne classification balancé

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

$$bal_{acc} = (TPR + TNR)/2$$

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

```
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
```

# Precision / Recall

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

$$precision = TP/(TP + FP)$$

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

$$recall = TP/(TP + FN)$$

```
from sklearn.metrics import precision_score, recall_sco
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-score

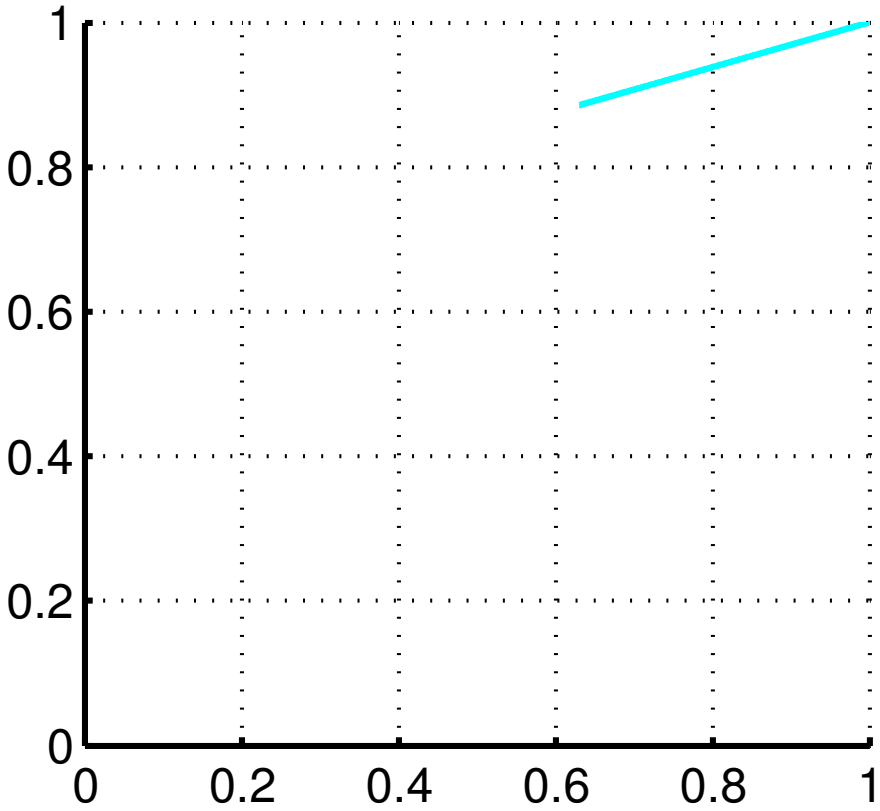
**F-score** : moyenne harmonique du recall et de la précision.

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

**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
```

# Courbes ROC et precision / recall




**ROC** (Receiver Operating Characteristics) et **precision/recall** : progrès de l'algorithme lorsqu'on fait varier le seuil de discrimination (valeur à partir de laquelle $\hat{y}$ vaut 1.