K plus proches voisins (K-NN)

K-Nearest Neighbors (K-NN)

Jérémie Cabessa

Laboratoire DAVID, UVSQ

Régression / Classification

- Dans le cadre de l'**apprentissage supervisé**, on distingue deux types de méthodes:
- **Méthodes de régression**

  La variable d'output (réponse) est **quantitative**.
- **Méthodes de classification**

  La variable d'output (réponse) est **qualitative**.

Régression / Classification

- Dans le cadre de l'**apprentissage supervisé**, on distingue deux types de méthodes:
- **Méthodes de régression**

  La variable d'output (réponse) est **quantitative**.
- **Méthodes de classification**

  La variable d'output (réponse) est **qualitative**.

Régression / Classification

- Dans le cadre de l'**apprentissage supervisé**, on distingue deux types de méthodes:
- **Méthodes de régression**

  La variable d'output (réponse) est **quantitative**.
- **Méthodes de classification**

  La variable d'output (réponse) est **qualitative**.

Classification

Ce chapitre présente une méthodes simple de classification:

- L'algorithme des **K plus proches voisins** ou (**K-Nearest Neighbors, K-NN**).
- K-NN peut être aisément généralisé dans un contexte de régression.

Classification

Ce chapitre présente une méthodes simple de classification:

- L'algorithme des **K plus proches voisins** ou (**K-Nearest Neighbors, K-NN**).
- K-NN peut être aisément généralisé dans un contexte de régression.

K plus proches voisins

- Soit un training set

$$S_{\text{train}} = \{(\boldsymbol{x_1}, y_1), \ldots, (\boldsymbol{x_N}, y_N)\}.$$

- Pour toute (nouvelle) observation x, on aimerait prédire sa classe y.
- L'algorithme **K plus proches voisins** procède ainsi:
  - On fixe le nombre de voisins $K \in \mathbb{N}$
  - Pour toute observation x, on prend les K plus proches voisins de x dans S_{train}
  - On prédit pour x la classe y qui apparaît le plus fréquemment dans ses K plus proches voisins.

K plus proches voisins

- Soit un training set

$$S_{\text{train}} = \{(\boldsymbol{x_1}, y_1), \ldots, (\boldsymbol{x_N}, y_N)\}.$$

- Pour toute (nouvelle) observation $\boldsymbol{x}$, on aimerait prédire sa classe y.
- L'algorithme **K plus proches voisins** procède ainsi:
  - On fixe le nombre de voisins $k \in \mathbb{N}$.
  - Pour toute observation $\boldsymbol{x}$, on prend les k plus proches voisins de $\boldsymbol{x}$ dans S_{train}.
  - On prédit pour $\boldsymbol{x}$ la classe y qui apparaît le plus fréquemment dans ses k plus proches voisins.

K plus proches voisins

- Soit un training set

$$S_{\text{train}} = \{(\boldsymbol{x_1}, y_1), \ldots, (\boldsymbol{x_N}, y_N)\}.$$

- Pour toute (nouvelle) observation $\boldsymbol{x}$, on aimerait prédire sa classe y.
- L'algorithme **K plus proches voisins** procède ainsi:
  - On fixe le nombre de voisins $k \in \mathbb{N}$.
  - Pour toute observation $\boldsymbol{x}$, on prend ses K plus proches voisins $(\boldsymbol{x_{i_1}}, y_{i_1}), \ldots, (\boldsymbol{x_{i_k}}, y_{i_k})$:
  - On prédit pour $\boldsymbol{x}$ la classe y qui apparaît le plus fréquemment dans ses k plus proches voisins.

K plus proches voisins

- Soit un training set

$$S_{\text{train}} = \{(\boldsymbol{x_1}, y_1), \ldots, (\boldsymbol{x_N}, y_N)\}.$$

- Pour toute (nouvelle) observation $\boldsymbol{x}$, on aimerait prédire sa classe y.
- L'algorithme **K plus proches voisins** procède ainsi:
  - On fixe le nombre de voisins $k \in \mathbb{N}$.
  - Pour toute observation $\boldsymbol{x}$, on prend ses K plus proches voisins $(\boldsymbol{x_{i_1}}, y_{i_1}), \ldots, (\boldsymbol{x_{i_k}}, y_{i_k})$:
  - On prédit pour $\boldsymbol{x}$ la classe y qui apparaît le plus fréquemment dans ses k plus proches voisins.

K plus proches voisins

- Soit un training set

$$S_{\text{train}} = \{(\boldsymbol{x_1}, y_1), \ldots, (\boldsymbol{x_N}, y_N)\}.$$

- Pour toute (nouvelle) observation $\boldsymbol{x}$, on aimerait prédire sa classe y.
- L'algorithme **K plus proches voisins** procède ainsi:
  - On fixe le nombre de voisins $k \in \mathbb{N}$.
  - Pour toute observation $\boldsymbol{x}$, on prend ses K plus proches voisins $(\boldsymbol{x_{i_1}}, y_{i_1}), \ldots, (\boldsymbol{x_{i_k}}, y_{i_k})$:
  - On prédit pour x la classe y qui apparaît le plus fréquemment dans ses k plus proches voisins.

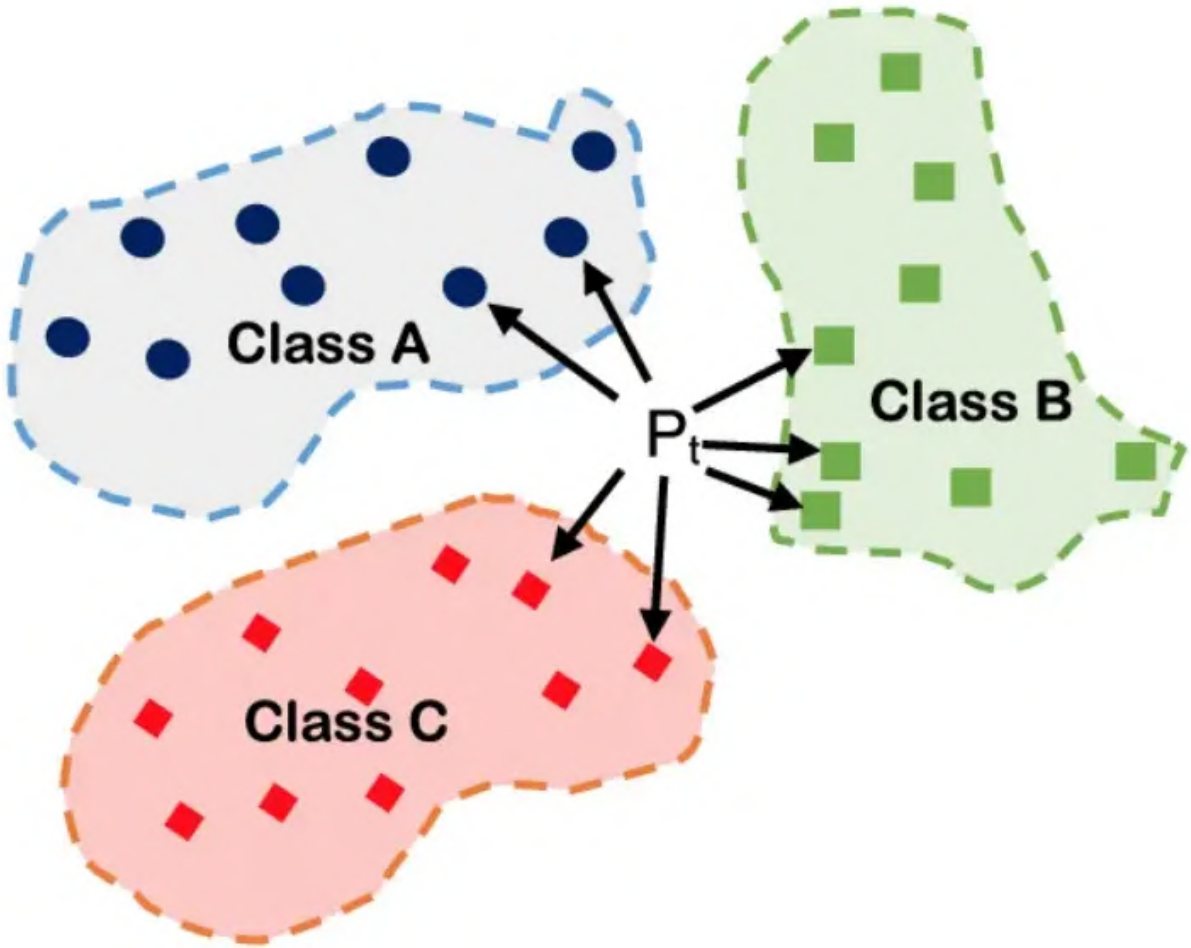
K plus proches voisins

- Soit un training set

$$S_{\text{train}} = \big\{(\boldsymbol{x_1}, y_1), \ldots, (\boldsymbol{x_N}, y_N)\big\}.$$

- Pour toute (nouvelle) observation $\boldsymbol{x}$, on aimerait prédire sa classe y.
- L'algorithme **K plus proches voisins** procède ainsi:
  - On fixe le nombre de voisins $k \in \mathbb{N}$.
  - Pour toute observation $\boldsymbol{x}$, on prend ses K plus proches voisins $(\boldsymbol{x_{i_1}}, y_{i_1}), \ldots, (\boldsymbol{x_{i_k}}, y_{i_k})$:
  - On prédit pour $\boldsymbol{x}$ la classe y qui apparaît le plus fréquemment dans ses k plus proches voisins.

K plus proches voisins



K plus proches voisins

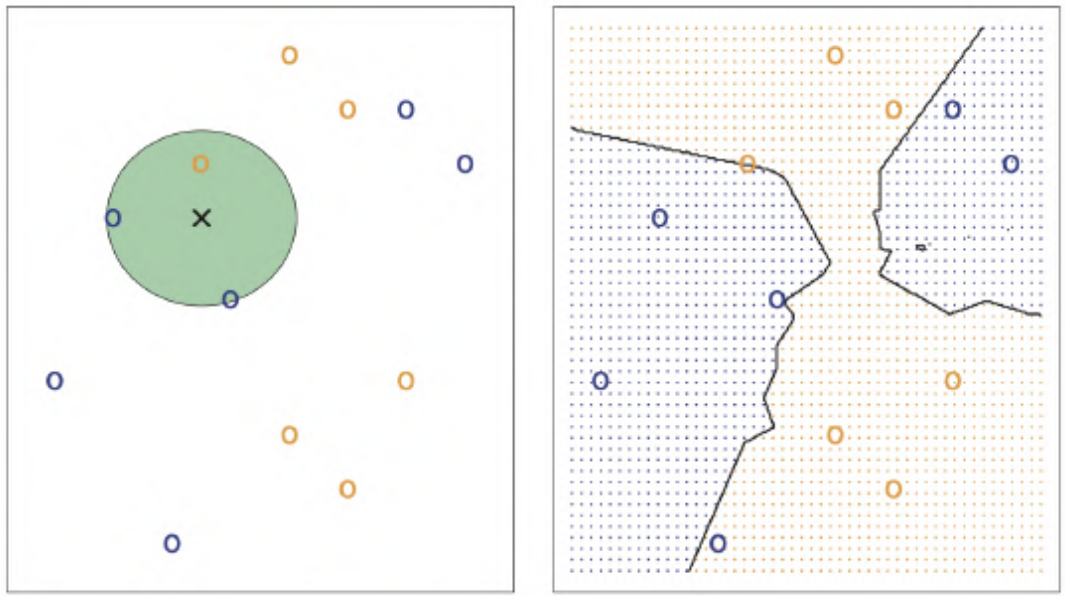


FIGURE 2.14. *The KNN approach, using $K = 3$, is illustrated in a simple situation with six blue observations and six orange observations.* Left: *a test observation at which a predicted class label is desired is shown as a black cross. The three closest points to the test observation are identified, and it is predicted that the test observation belongs to the most commonly-occurring class, in this case blue.* Right: *The KNN decision boundary for this example is shown in black. The blue grid indicates the region in which a test observation will be assigned to the blue class, and the orange grid indicates the region in which it will be assigned to the orange class.*

K plus proches voisins

- Plus K est petit, plus la frontière de décision est non-linaire:
  $\rightarrow$ petit biais mais grande variance (bias-variance trade-off)
- Plus K est grand, plus la frontière de décision est linaire:
  $\rightarrow$ petite variance mais grand biais (bias-variance trade-off)

K plus proches voisins

- Plus K est petit, plus la frontière de décision est non-linaire:
  $\rightarrow$ petit biais mais grande variance (bias-variance trade-off)
- Plus K est grand, plus la frontière de décision est linaire:
  $\rightarrow$ petite variance mais grand biais (bias-variance trade-off)

K plus proches voisins

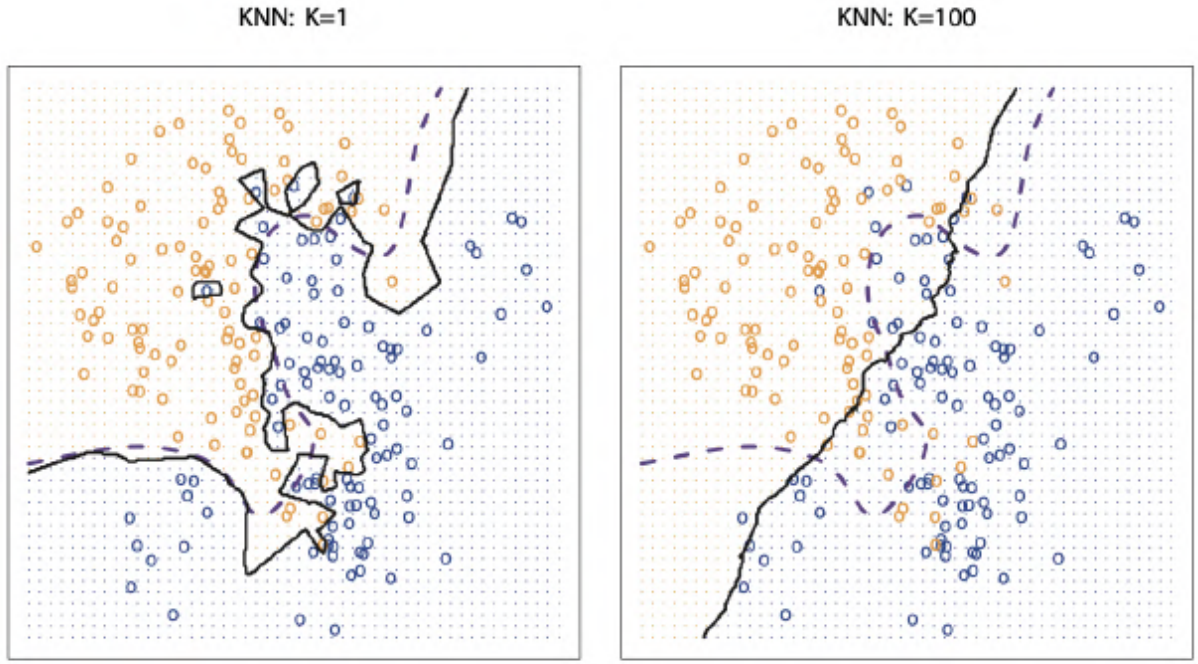


FIGURE 2.16. *A comparison of the KNN decision boundaries (solid black curves) obtained using $K = 1$ and $K = 100$ on the data from Figure 2.13. With $K = 1$, the decision boundary is overly flexible, while with $K = 100$ it is not sufficiently flexible. The Bayes decision boundary is shown as a purple dashed line.*

K plus proches voisins

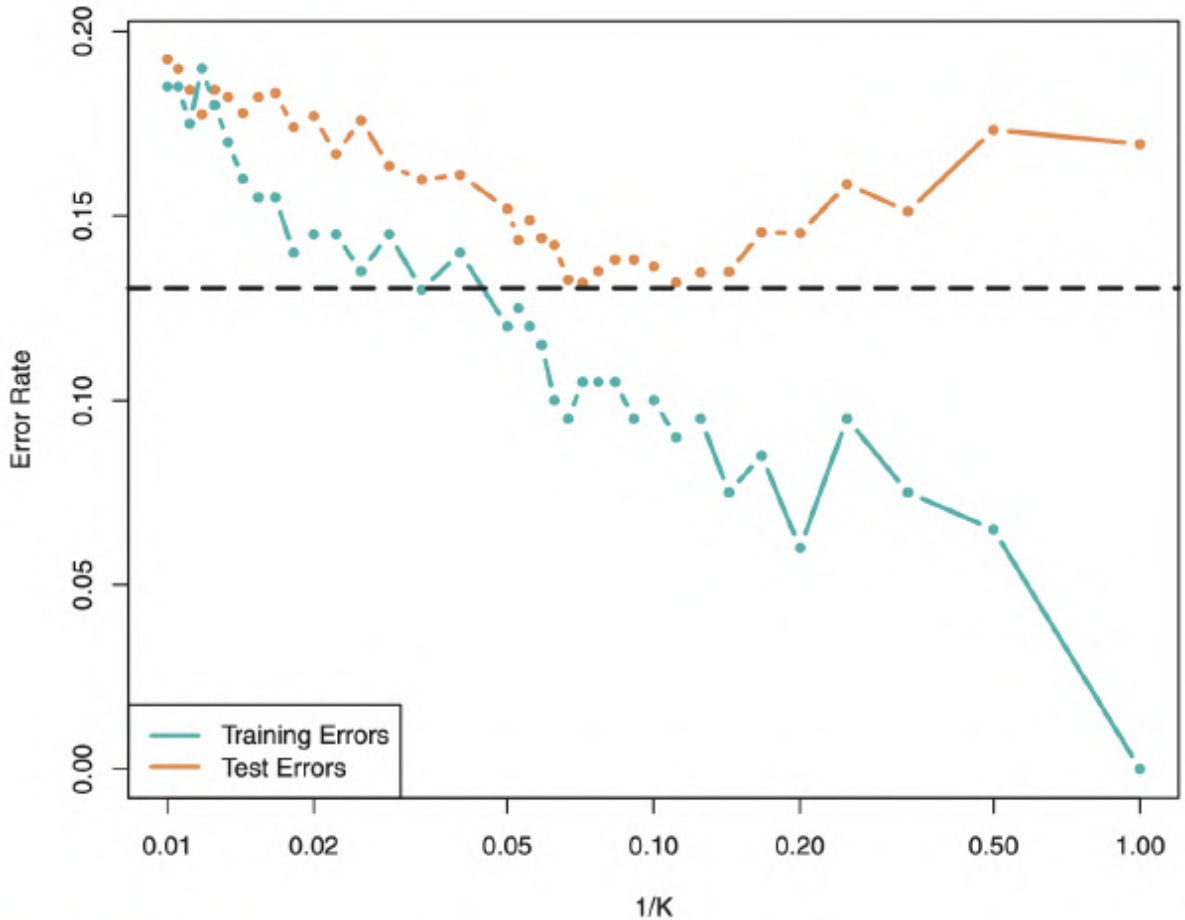


FIGURE 2.17. *The KNN training error rate (blue, 200 observations) and test error rate (orange, 5,000 observations) on the data from Figure 2.13, as the level of flexibility (assessed using $1/K$) increases, or equivalently as the number of neighbors K decreases. The black dashed line indicates the Bayes error rate. The jumpiness of the curves is due to the small size of the training data set.*

K plus proches voisins

- Soient $\boldsymbol{X} = (X_1, \ldots, X_p)$ des variables explicatives et Y une variable réponse qualitative (C classes différentes).
- On codes les réalisations de Y par $1, 2, \ldots, C$.
- Pour toute classe $c = 1, \ldots, C$, on aimerait modéliser les probabilités que $Y = c$ étant données une réalisation des variables $\boldsymbol{X} = X_1, \ldots, X_p$:

$$\Pr(Y = c \mid \boldsymbol{X})$$

- Pour toute observation $\boldsymbol{x} = (x_1, \ldots, x_p)$, on prédit la classe $\hat{c}$ associée à la plus grande probabilité:

$$\hat{c} = \arg\max_{c \in C} \Pr(Y = c \mid \boldsymbol{X} = \boldsymbol{x})$$

K plus proches voisins

- Soient $\boldsymbol{X} = (X_1, \ldots, X_p)$ des variables explicatives et Y une variable réponse qualitative (C classes différentes).
- On codes les réalisations de Y par $1, 2, \ldots, C$.
- Pour toute classe $c = 1, \ldots, C$, on aimerait modéliser les probabilités que $Y = c$ étant données une réalisation des variables $\boldsymbol{X} = X_1, \ldots, X_p$:

$$\Pr(Y = c \mid \boldsymbol{X})$$

- Pour toute observation $\boldsymbol{x} = (x_1, \ldots, x_p)$, on prédit la classe $\hat{c}$ associée à la plus grande probabilité:

$$\hat{c} = \arg\max_{c \in C} \Pr(Y = c \mid \boldsymbol{X} = \boldsymbol{x})$$

K plus proches voisins

- Soient $\boldsymbol{X} = (X_1, \ldots, X_p)$ des variables explicatives et Y une variable réponse qualitative (C classes différentes).
- On codes les réalisations de Y par $1, 2, \ldots, C$.
- Pour toute classe $c = 1, \ldots, C$, on aimerait modéliser les probabilités que $Y = c$ étant données une réalisation des variables $\boldsymbol{X} = X_1, \ldots, X_p$:

$$\Pr(Y = c \mid \boldsymbol{X})$$

- Pour toute observation $\boldsymbol{x} = (x_1, \ldots, x_p)$, on prédit la classe $\hat{c}$ associée à la plus grande probabilité:

$$\hat{c} = \arg\max_{c \in C} \Pr(Y = c \mid \boldsymbol{X} = \boldsymbol{x})$$

K plus proches voisins

- Soient $\boldsymbol{X} = (X_1, \ldots, X_p)$ des variables explicatives et Y une variable réponse qualitative (C classes différentes).
- On codes les réalisations de Y par $1, 2, \ldots, C$.
- Pour toute classe $c = 1, \ldots, C$, on aimerait modéliser les probabilités que $Y = c$ étant données une réalisation des variables $\boldsymbol{X} = X_1, \ldots, X_p$:

$$\Pr(Y = c \mid \boldsymbol{X})$$

- Pour toute observation $\boldsymbol{x} = (x_1, \ldots, x_p)$, on prédit la classe $\hat{c}$ associée à la plus grande probabilité:

$$\hat{c} = \arg\max_{c \in C} \Pr(Y = c \mid \boldsymbol{X} = \boldsymbol{x})$$

K plus proches voisins

- Soit un training set

$$S_{\text{train}} = \big\{(\boldsymbol{x_1}, y_1), \ldots, (\boldsymbol{x_N}, y_N)\big\}.$$

- L'algorithme **K-Nearest Neighbors** procède ainsi: pour toute (nouvelle) observation $\boldsymbol{x} = (x_1, \ldots, x_p)$, on estime $\Pr(Y = c \mid \boldsymbol{X} = \boldsymbol{x})$ par:

$$\hat{\Pr}(Y = c \mid \boldsymbol{X} = \boldsymbol{x}) = \frac{1}{K} \sum_{\boldsymbol{x_i} \in \mathcal{N}_{\boldsymbol{x}}^K} \mathrm{I}(y_i = c)$$

où $\mathcal{N}_{\boldsymbol{x}}^K$ désigne les K plus proches voisins de $\boldsymbol{x}$ et $\mathrm{I}(.)$ la fonction indicatrice (vaut 1 si $y_i = c$ et 0 sinon).

- Pour $\boldsymbol{x}$, on prédit la classe $\hat{c}$ donnée par:

$$\hat{c} = \arg\max_{c \in C} \hat{\Pr}(Y = c \mid \boldsymbol{X} = \boldsymbol{x})$$

K plus proches voisins

- Soit un training set

$$S_{\text{train}} = \big\{(\boldsymbol{x_1}, y_1), \ldots, (\boldsymbol{x_N}, y_N)\big\}.$$

- L'algorithme **K-Nearest Neighbors** procède ainsi: pour toute (nouvelle) observation $\boldsymbol{x} = (x_1, \ldots, x_p)$, on estime $\Pr(Y = c \mid \boldsymbol{X} = \boldsymbol{x})$ par:

$$\hat{\Pr}(Y = c \mid \boldsymbol{X} = \boldsymbol{x}) = \frac{1}{K} \sum_{\boldsymbol{x_i} \in \mathcal{N}_{\boldsymbol{x}}^K} \mathrm{I}(y_i = c)$$

où $\mathcal{N}_{\boldsymbol{x}}^K$ désigne les K plus proches voisins de $\boldsymbol{x}$ et $\mathrm{I}(.)$ la fonction indicatrice (vaut 1 si $y_i = c$ et 0 sinon).

- Pour $\boldsymbol{x}$, on prédit la classe $\hat{c}$ donnée par:

$$\hat{c} = \arg\max_{c \in C} \hat{\Pr}(Y = c \mid \boldsymbol{X} = \boldsymbol{x})$$

K plus proches voisins

- Soit un training set

$$S_{\text{train}} = \big\{(\boldsymbol{x_1}, y_1), \ldots, (\boldsymbol{x_N}, y_N)\big\}.$$

- L'algorithme **K-Nearest Neighbors** procède ainsi: pour toute (nouvelle) observation $\boldsymbol{x} = (x_1, \ldots, x_p)$, on estime $\Pr(Y = c \mid \boldsymbol{X} = \boldsymbol{x})$ par:

$$\hat{\Pr}(Y = c \mid \boldsymbol{X} = \boldsymbol{x}) = \frac{1}{K} \sum_{\boldsymbol{x_i} \in \mathcal{N}_{\boldsymbol{x}}^K} \mathrm{I}(y_i = c)$$

où $\mathcal{N}_{\boldsymbol{x}}^K$ désigne les K plus proches voisins de $\boldsymbol{x}$ et $\mathrm{I}(.)$ la fonction indicatrice (vaut 1 si $y_i = c$ et 0 sinon).

- Pour $\boldsymbol{x}$, on prédit la classe $\hat{c}$ donnée par:

$$\hat{c} = \arg\max_{c \in C} \hat{\Pr}(Y = c \mid \boldsymbol{X} = \boldsymbol{x})$$

K-NN

Algorithm 1: knn(dataset, k, x_{new})

```
dist_and_targets = []
for x_i, y_i in dataset do
    d_i = distance(x_i, x_new)
    dist_and_targets.append([d_i, y_i])
end
dist_and_targets = sort_by_first_component(dist_and_targets)
y = most_frequent_target(dist_and_targets[0 : k])
return y
```

K-NN

Algorithm 1: knn(dataset, k, x_{new})

```
dist_and_targets = []
for x_i, y_i in dataset do
    d_i = distance(x_i, x_new)
    dist_and_targets.append([d_i, y_i])
end
dist_and_targets = sort_by_first_component(dist_and_targets)
y = most_frequent_target(dist_and_targets[0 : k])
return y
```

K-NN

Algorithm 1: knn(dataset, k, x_{new})

```
dist_and_targets = []
for x_i, y_i in dataset do
    d_i = distance(x_i, x_new)
    dist_and_targets.append([d_i, y_i])
end
dist_and_targets = sort_by_first_component(dist_and_targets)
y = most_frequent_target(dist_and_targets[0 : k])
return y
```

K-NN

Algorithm 1: knn(dataset, k, x_{new})

```
dist_and_targets = []
for x_i, y_i in dataset do
    d_i = distance(x_i, x_new)
    dist_and_targets.append([d_i, y_i])
end
dist_and_targets = sort_by_first_component(dist_and_targets)
y = most_frequent_target(dist_and_targets[0 : k])
return y
```

K-NN

Algorithm 1: knn(dataset, k, x_{new})

```
dist_and_targets = []
for x_i, y_i in dataset do
    d_i = distance(x_i, x_new)
    dist_and_targets.append([d_i, y_i])
end
dist_and_targets = sort_by_first_component(dist_and_targets)
y = most_frequent_target(dist_and_targets[0 : k])
return y
```

K-NN

Algorithm 1: knn(dataset, k, x_{new})

```
dist_and_targets = []
for x_i, y_i in dataset do
    d_i = distance(x_i, x_new)
    dist_and_targets.append([d_i, y_i])
end
dist_and_targets = sort_by_first_component(dist_and_targets)
y = most_frequent_target(dist_and_targets[0 : k])
return y
```

K plus proches voisins

- On peut facilement généraliser l'algorithme K-NN pour un contexte de régression.
- Pour toute observation x, au lieu de lui prédire la classe $\hat{c}$ qui apparaît le plus souvent parmi les classes c_i des K plus proches voisins...
- on lui prédit la valeur moyenne $\hat{y}$ des targets y_i de ses K plus proches voisins:

$$\hat{y} = \frac{1}{K} \sum_{x_i \in \mathcal{N}_x^K} y_i$$

K plus proches voisins

- On peut facilement généraliser l'algorithme K-NN pour un contexte de régression.
- Pour toute observation $\boldsymbol{x}$, au lieu de lui prédire la classe $\hat{c}$ qui apparaît le plus souvent parmi les classes c_i des K plus proches voisins...
- on lui prédit la valeur moyenne $\hat{y}$ des targets y_i de ses K plus proches voisins:

$$\hat{y} = \frac{1}{K} \sum_{\boldsymbol{x_i} \in \mathcal{N}_{\boldsymbol{x}}^K} y_i$$

K plus proches voisins

- On peut facilement généraliser l'algorithme K-NN pour un contexte de régression.
- Pour toute observation $\boldsymbol{x}$, au lieu de lui prédire la classe $\hat{c}$ qui apparaît le plus souvent parmi les classes c_i des K plus proches voisins...
- on lui prédit la valeur moyenne $\hat{y}$ des targets y_i de ses K plus proches voisins:

$$\hat{y} = \frac{1}{K} \sum_{\boldsymbol{x_i} \in \mathcal{N}_{\boldsymbol{x}}^K} y_i$$

Matrice de confusion

- Comment évaluer la performance d'un classifieur binaire?
- On utilise les **matrices de confusion** et les métriques qui en découlent.
- **Exemple:** Soit un groupe de 1000 personnes dans lequel se trouve 10 individus dangereux.
- Supposons qu'on ait entraîné qui modèle (classifieur binaire) qui prédise si un individu est dangereux ou non.
- Mais ce modèle est très défaillant: parmi 10 individus dangereux, seul 1 est détecté comme dangereux; sinon, parmi les 990 personnes inoffensives, toutes sont détectées comme inoffensives.

Matrice de confusion

- Comment évaluer la performance d'un classifieur binaire?
- On utilise les **matrices de confusion** et les métriques qui en découlent.
- **Exemple:** Soit un groupe de 1000 personnes dans lequel se trouve 10 individus dangereux.
- Supposons qu'on ait entraîné qui modèle (classifieur binaire) qui prédise si un individu est dangereux ou non.
- Mais ce modèle est très défaillant: parmi 10 individus dangereux, seul 1 est détecté comme dangereux; sinon, parmi les 990 personnes inoffensives, toutes sont détectées comme inoffensives.

Matrice de confusion

- Comment évaluer la performance d'un classifieur binaire?
- On utilise les **matrices de confusion** et les métriques qui en découlent.
- **Exemple:** Soit un groupe de 1000 personnes dans lequel se trouve 10 individus dangereux.
- Supposons qu'on ait entraîné qui modèle (classifieur binaire) qui prédise si un individu est dangereux ou non.
- Mais ce modèle est très défaillant: parmi 10 individus dangereux, seul 1 est détecté comme dangereux; sinon, parmi les 990 personnes inoffensives, toutes sont détectées comme inoffensives.

Matrice de confusion

- Comment évaluer la performance d'un classifieur binaire?
- On utilise les **matrices de confusion** et les métriques qui en découlent.
- **Exemple:** Soit un groupe de 1000 personnes dans lequel se trouve 10 individus dangereux.
- Supposons qu'on ait entraîné qui modèle (classifieur binaire) qui prédise si un individu est dangereux ou non.
- Mais ce modèle est très défaillant: parmi 10 individus dangereux, seul 1 est détecté comme dangereux; sinon, parmi les 990 personnes inoffensives, toutes sont détectées comme inoffensives.

Matrice de confusion

- Comment évaluer la performance d'un classifieur binaire?
- On utilise les **matrices de confusion** et les métriques qui en découlent.
- **Exemple:** Soit un groupe de 1000 personnes dans lequel se trouve 10 individus dangereux.
- Supposons qu'on ait entraîné qui modèle (classifieur binaire) qui prédise si un individu est dangereux ou non.
- Mais ce modèle est très défaillant: parmi 10 individus dangereux, seul 1 est détecté comme dangereux; sinon, parmi les 990 personnes inoffensives, toutes sont détectées comme inoffensives.

Matrice de confusion

- Les résultats du classifieur sont les suivants:

		prédiction	
		inoffensif	dangereux
réalité	inoffensif	990	0
	dangereux	9	1

- L'erreur totale du modèle n'est que de $\frac{9+0}{1000} = 0.9\%$.
- Autrement dit, l'accuracy du modèle est de $\frac{990+1}{1000} = 99.1\%$.
- Pourtant, le classifieur est mauvais: il laisse passer presque tous les dangereux.
- C'est le **paradoxe de l'accuracy**.

Matrice de confusion

- Les résultats du classifieur sont les suivants:

		prédiction	
		inoffensif	dangereux
réalité	inoffensif	990	0
	dangereux	9	1

- L'erreur totale du modèle n'est que de $\frac{9+0}{1000} = 0.9\%$.
- Autrement dit, l'accuracy du modèle est de $\frac{990+1}{1000} = 99.1\%$.
- Pourtant, le classifieur est mauvais: il laisse passer presque tous les dangereux.
- C'est le paradoxe de l'accuracy.

Matrice de confusion

- Les résultats du classifieur sont les suivants:

		prédiction	
		inoffensif	dangereux
réalité	inoffensif	990	0
	dangereux	9	1

- L'erreur totale du modèle n'est que de $\frac{9+0}{1000} = 0.9\%$.
- Autrement dit, l'accuracy du modèle est de $\frac{990+1}{1000} = 99.1\%$.
- Pourtant, le classifieur est mauvais: il laisse passer presque tous les dangereux.
- C'est le **paradoxe de l'accuracy**.

Matrice de confusion

- Les résultats du classifieur sont les suivants:

		prédiction	
		inoffensif	dangereux
réalité	inoffensif	990	0
	dangereux	9	1

- L'erreur totale du modèle n'est que de $\frac{9+0}{1000} = 0.9\%$.
- Autrement dit, l'accuracy du modèle est de $\frac{990+1}{1000} = 99.1\%$.
- Pourtant, le classifieur est mauvais: il laisse passer presque tous les dangereux.
- C'est le paradoxe de l'accuracy.

Matrice de confusion

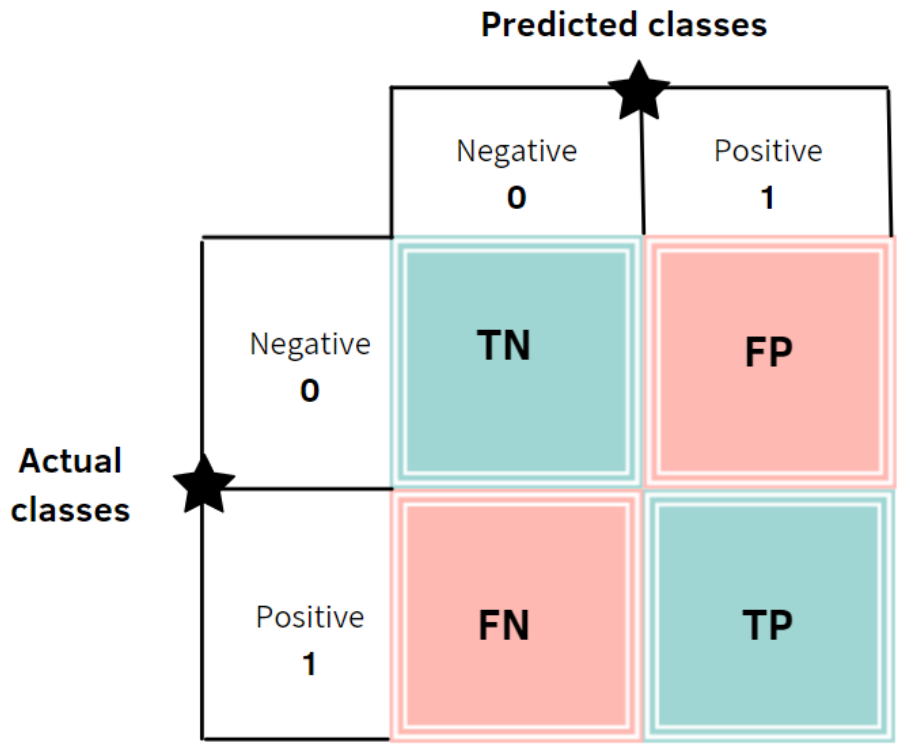
- Les résultats du classifieur sont les suivants:

		prédiction	
		inoffensif	dangereux
réalité	inoffensif	990	0
	dangereux	9	1

- L'erreur totale du modèle n'est que de $\frac{9+0}{1000} = 0.9\%$.
- Autrement dit, l'accuracy du modèle est de $\frac{990+1}{1000} = 99.1\%$.
- Pourtant, le classifieur est mauvais: il laisse passer presque tous les dangereux.
- C'est le **paradoxe de l'accuracy**.

Matrice de confusion

- De manière générale, la **matrice de confusion** d'un classifieur est la suivante:



Matrice de confusion

- **True positive:** prédit comme positif (positive), et c'est juste dans la réalité (True).
- **True negative:** prédit comme négatif (negative), et c'est juste dans la réalité (True).
- **False positive:** prédit comme positif (positive), et c'est faux dans la réalité (False).
- **False negative:** prédit comme négatif (negative), et c'est faux dans la réalité (False).

Matrice de confusion

- **True positive:** prédit comme positif (positive), et c'est juste dans la réalité (True).
- **True negative:** prédit comme négatif (negative), et c'est juste dans la réalité (True).
- **False positive:** prédit comme positif (positive), et c'est faux dans la réalité (False).
- **False negative:** prédit comme négatif (negative), et c'est faux dans la réalité (False).

Matrice de confusion

- **True positive:** prédit comme positif (positive), et c'est juste dans la réalité (True).
- **True negative:** prédit comme négatif (negative), et c'est juste dans la réalité (True).
- **False positive:** prédit comme positif (positive), et c'est faux dans la réalité (False).
- **False negative:** prédit comme négatif (negative), et c'est faux dans la réalité (False).

Matrice de confusion

- **True positive:** prédit comme positif (positive), et c'est juste dans la réalité (True).
- **True negative:** prédit comme négatif (negative), et c'est juste dans la réalité (True).
- **False positive:** prédit comme positif (positive), et c'est faux dans la réalité (False).
- **False negative:** prédit comme négatif (negative), et c'est faux dans la réalité (False).

Matrice de confusion

- **Accuracy**: $Accuracy = \frac{TP+TN}{N+P}$
- False positive rate: $FPR = \frac{FP}{N}$
- True positive rate / Recall: $TPR = Recall = \frac{TP}{P}$
- Precision: $Precision = \frac{TP}{P^*}$
- Negative prediction value : $NPV = \frac{TN}{N^*}$

		Predicted class		
		− or Null	+ or Non-null	Total
True class	− or Null	True Neg. (TN)	False Pos. (FP)	N
	+ or Non-null	False Neg. (FN)	True Pos. (TP)	P
	Total	N*	P*	

TABLE 4.6. *Possible results when applying a classifier or diagnostic test to a population.*

Matrice de confusion

- **Accuracy**: $Accuracy = \frac{TP+TN}{N+P}$
- **False positive rate**: $FPR = \frac{FP}{N}$
- **True positive rate / Recall**: $TPR = Recall = \frac{TP}{P}$
- **Precision**: $Precision = \frac{TP}{P^*}$
- **Negative prediction value** : $NPV = \frac{TN}{N^*}$

		Predicted class		
		− or Null	+ or Non-null	Total
True class	− or Null	True Neg. (TN)	False Pos. (FP)	N
	+ or Non-null	False Neg. (FN)	True Pos. (TP)	P
	Total	N*	P*	

TABLE 4.6. *Possible results when applying a classifier or diagnostic test to a population.*

Matrice de confusion

- **Accuracy**: $Accuracy = \frac{TP+TN}{N+P}$
- **False positive rate**: $FPR = \frac{FP}{N}$
- **True positive rate / Recall**: $TPR = Recall = \frac{TP}{P}$
- **Precision**: $Precision = \frac{TP}{P^*}$
- **Negative prediction value** : $NPV = \frac{TN}{N^*}$

		Predicted class		
		− or Null	+ or Non-null	Total
True class	− or Null	True Neg. (TN)	False Pos. (FP)	N
	+ or Non-null	False Neg. (FN)	True Pos. (TP)	P
	Total	N*	P*	

TABLE 4.6. *Possible results when applying a classifier or diagnostic test to a population.*

Matrice de confusion

- **Accuracy**: $Accuracy = \frac{TP+TN}{N+P}$
- **False positive rate**: $FPR = \frac{FP}{N}$
- **True positive rate / Recall**: $TPR = Recall = \frac{TP}{P}$
- **Precision**: $Precision = \frac{TP}{P^*}$
- Negative prediction value : $NPV = \frac{TN}{N^*}$

		Predicted class		
		− or Null	+ or Non-null	Total
True class	− or Null	True Neg. (TN)	False Pos. (FP)	N
	+ or Non-null	False Neg. (FN)	True Pos. (TP)	P
	Total	N*	P*	

TABLE 4.6. *Possible results when applying a classifier or diagnostic test to a population.*

Matrice de confusion

- **Accuracy**: $Accuracy = \frac{TP+TN}{N+P}$
- **False positive rate**: $FPR = \frac{FP}{N}$
- **True positive rate / Recall**: $TPR = Recall = \frac{TP}{P}$
- **Precision**: $Precision = \frac{TP}{P^*}$
- **Negative prediction value** : $NPV = \frac{TN}{N^*}$

		Predicted class		
		− or Null	+ or Non-null	Total
True class	− or Null	True Neg. (TN)	False Pos. (FP)	N
	+ or Non-null	False Neg. (FN)	True Pos. (TP)	P
	Total	N*	P*	

TABLE 4.6. *Possible results when applying a classifier or diagnostic test to a population.*

Matrice de confusion

Dans notre example, on a:

- $Accuracy = \frac{990+1}{1000} = 99.1$ (bon)
- $FPR = \frac{0}{990+0} = 0\%$ (bon, low=good)
- $Recall = \frac{TP}{P} = \frac{1}{9+1} = 10\%$: (mauvais!)
- $Precision = \frac{TP}{P^*} = \frac{1}{0+1} = 100\%$ (bon)
- $\frac{TN}{N^*} = \frac{990}{990+9} = 99.099\%$ (bon)
- Dans ce cas, les bonnes performances viennent du fait que les 2 classes sont fortement "imbalanced". Toutefois, le *recall* nous indique que notre modèle est mauvais en certains aspects.

		prédiction	
		inoffensif	dangereux
réalité	inoffensif	990	0
	dangereux	9	1

Matrice de confusion

Dans notre example, on a:

- $Accuracy = \frac{990+1}{1000} = 99.1$ (bon)
- $FPR = \frac{0}{990+0} = 0\%$ (bon, low=good)
- $Recall = \frac{TP}{P} = \frac{1}{9+1} = 10\%$: (mauvais!)
- $Precision = \frac{TP}{P^*} = \frac{1}{0+1} = 100\%$ (bon)
- $\frac{TN}{N^*} = \frac{990}{990+9} = 99.099\%$ (bon)
- Dans ce cas, les bonnes performances viennent du fait que les 2 classes sont fortement "imbalanced". Toutefois, le *recall* nous indique que notre modèle est mauvais en certains aspects.

		prédiction	
		inoffensif	dangereux
réalité	inoffensif	990	0
	dangereux	9	1

Matrice de confusion

Dans notre example, on a:

- $Accuracy = \frac{990+1}{1000} = 99.1$ (bon)
- $FPR = \frac{0}{990+0} = 0\%$ (bon, low=good)
- $Recall = \frac{TP}{P} = \frac{1}{9+1} = 10\%$: (mauvais!)
- $Precision = \frac{TP}{P^*} = \frac{1}{0+1} = 100\%$ (bon)
- $\frac{TN}{N^*} = \frac{990}{990+9} = 99.099\%$ (bon)
- Dans ce cas, les bonnes performances viennent du fait que les 2 classes sont fortement "imbalanced". Toutefois, le *recall* nous indique que notre modèle est mauvais en certains aspects.

		prédiction	
		inoffensif	dangereux
réalité	inoffensif	990	0
	dangereux	9	1

Matrice de confusion

Dans notre example, on a:

- $Accuracy = \frac{990+1}{1000} = 99.1$ (bon)
- $FPR = \frac{0}{990+0} = 0\%$ (bon, low=good)
- $Recall = \frac{TP}{P} = \frac{1}{9+1} = 10\%$: (mauvais!)
- $Precision = \frac{TP}{P^*} = \frac{1}{0+1} = 100\%$ (bon)
- $\frac{TN}{N^*} = \frac{990}{990+9} = 99.099\%$ (bon)
- Dans ce cas, les bonnes performances viennent du fait que les 2 classes sont fortement "imbalanced". Toutefois, le *recall* nous indique que notre modèle est mauvais en certains aspects.

		prédiction	
		inoffensif	dangereux
réalité	inoffensif	990	0
	dangereux	9	1

Matrice de confusion

Dans notre example, on a:

- $Accuracy = \frac{990+1}{1000} = 99.1$ (bon)
- $FPR = \frac{0}{990+0} = 0\%$ (bon, low=good)
- $Recall = \frac{TP}{P} = \frac{1}{9+1} = 10\%$: (mauvais!)
- $Precision = \frac{TP}{P^*} = \frac{1}{0+1} = 100\%$ (bon)
- $\frac{TN}{N^*} = \frac{990}{990+9} = 99.099\%$ (bon)
- Dans ce cas, les bonnes performances viennent du fait que les 2 classes sont fortement "imbalanced". Toutefois, le *recall* nous indique que notre modèle est mauvais en certains aspects.

		prédiction	
		inoffensif	dangereux
réalité	inoffensif	990	0
	dangereux	9	1

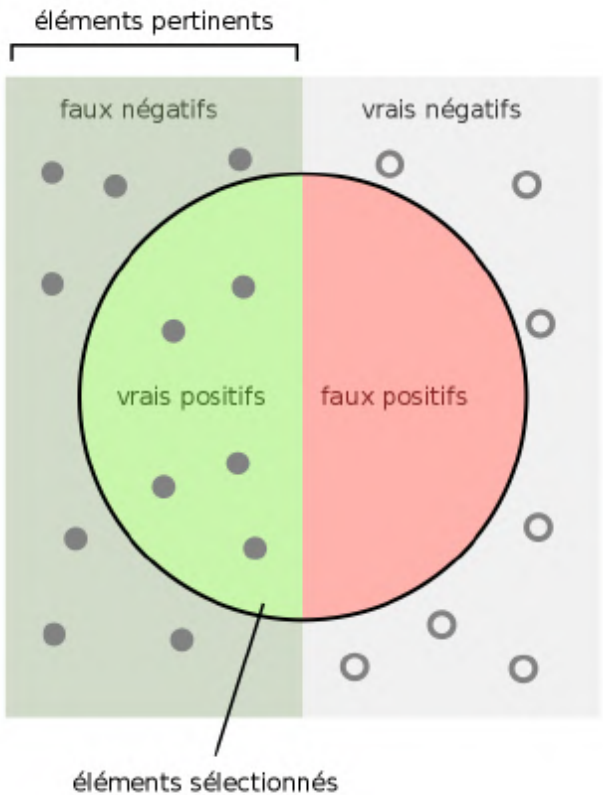
Matrice de confusion

Dans notre example, on a:

- $Accuracy = \frac{990+1}{1000} = 99.1$ (bon)
- $FPR = \frac{0}{990+0} = 0\%$ (bon, low=good)
- $Recall = \frac{TP}{P} = \frac{1}{9+1} = 10\%$: (mauvais!)
- $Precision = \frac{TP}{P^*} = \frac{1}{0+1} = 100\%$ (bon)
- $\frac{TN}{N^*} = \frac{990}{990+9} = 99.099\%$ (bon)
- Dans ce cas, les bonnes performances viennent du fait que les 2 classes sont fortement "imbalanced". Toutefois, le *recall* nous indique que notre modèle est mauvais en certains aspects.

		prédiction	
		inoffensif	dangereux
réalité	inoffensif	990	0
	dangereux	9	1

Matrice de confusion



Combien de candidats sélectionnés sont pertinents ?

$$\text{Précision} = \frac{\text{vrais positifs}}{\text{éléments sélectionnés}}$$

Combien d'éléments pertinents sont sélectionnés ?

$$\text{Rappel} = \frac{\text{vrais positifs}}{\text{éléments pertinents}}$$

Matrice de confusion

- On rappel que notre classifieur retourne une probabilité

$$\hat{p} = \hat{\Pr}(Y = 1 \mid \boldsymbol{X} = x)$$

et qu'on classifie x dans la classe 1 si $\hat{p} \geq 0.5$ et dans la classe 0 sinon.

- Le choix du seuil (threshold) de 0.5 est arbitraire...
- On peut créer un classifieur plus ou moins sévères en augmentant ou diminuant ce seuil.

Matrice de confusion

- On rappel que notre classifieur retourne une probabilité

$$\hat{p} = \hat{\Pr}(Y = 1 \mid \boldsymbol{X} = x)$$

  et qu'on classifie x dans la classe 1 si $\hat{p} \geq 0.5$ et dans la classe 0 sinon.
- Le choix du seuil (threshold) de 0.5 est arbitraire...
- On peut créer un classifieur plus ou moins sévères en augmentant ou diminuant ce seuil.

Matrice de confusion

- On rappel que notre classifieur retourne une probabilité

$$\hat{p} = \hat{\Pr}(Y = 1 \mid \boldsymbol{X} = x)$$

  et qu'on classifie x dans la classe 1 si $\hat{p} \geq 0.5$ et dans la classe 0 sinon.
- Le choix du seuil (threshold) de 0.5 est arbitraire...
- On peut créer un classifieur plus ou moins sévères en augmentant ou diminuant ce seuil.

Matrice de confusion

- Voici deux matrices de confusions pour un calssifieur qui utilise les seuils de 0.5 et 0.2, respectivement.
- On voit que le nombre de prédictions positives augmente de 104 à 430 (puisque le seuil est plus bas).

		True default status		
		No	Yes	Total
Predicted default status	No	9,644	252	9,896
	Yes	23	81	104
	Total	9,667	333	10,000

		True default status		
		No	Yes	Total
Predicted default status	No	9,432	138	9,570
	Yes	235	195	430
	Total	9,667	333	10,000

Matrice de confusion

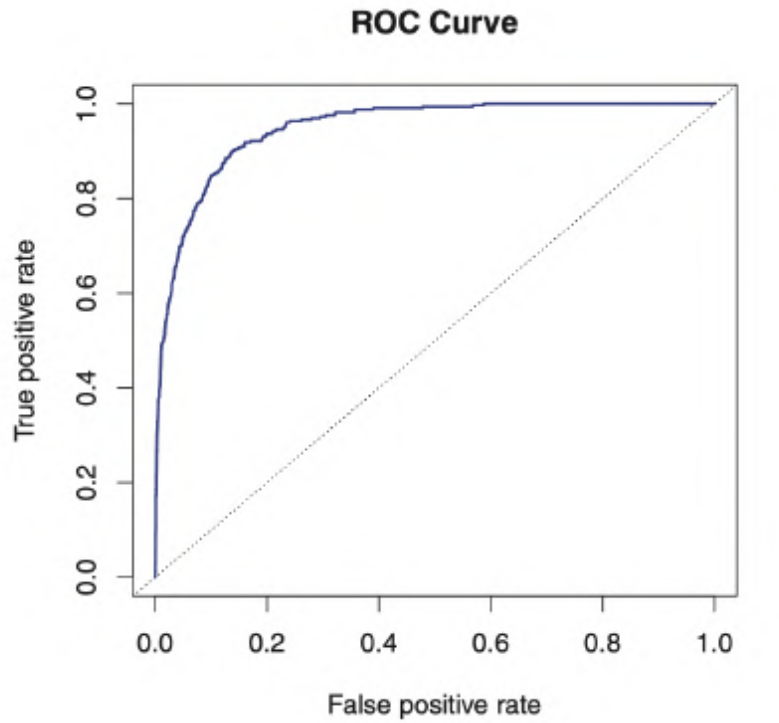
- Voici deux matrices de confusions pour un calssifieur qui utilise les seuils de 0.5 et 0.2, respectivement.
- On voit que le nombre de prédictions positives augmente de 104 à 430 (puisque le seuil est plus bas).

		True default status		
		No	Yes	Total
Predicted default status	No	9,644	252	9,896
	Yes	23	81	104
	Total	9,667	333	10,000

		True default status		
		No	Yes	Total
Predicted default status	No	9,432	138	9,570
	Yes	235	195	430
	Total	9,667	333	10,000

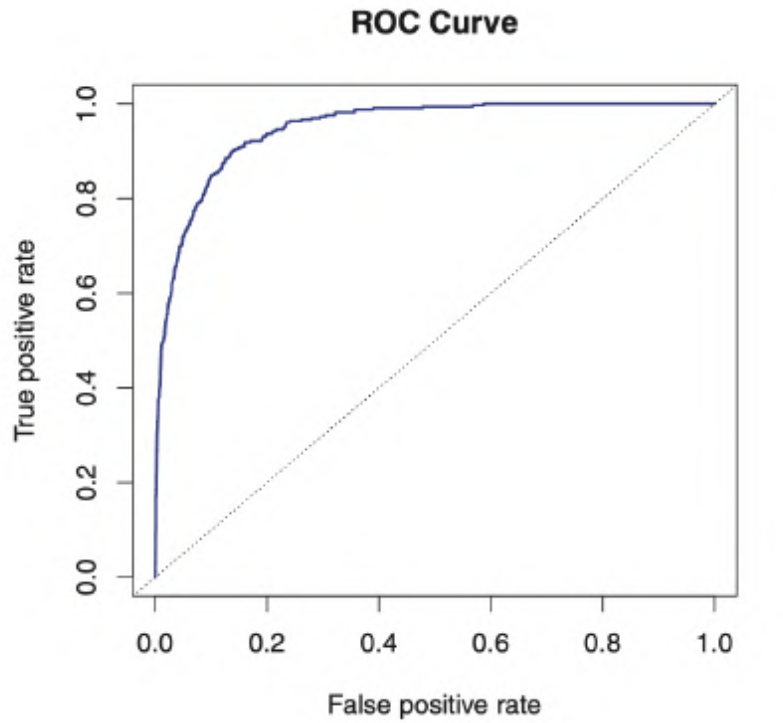
Matrice de confusion

- Un moyen de juger la performance d'un classifier est de faire sa **courbe ROC (ROC curve)**.
- C'est la courbe du "true positive rate / recall" en fonction du "false positive rate" paramétrée par le seuil θ (θ varie de 1 à 0).



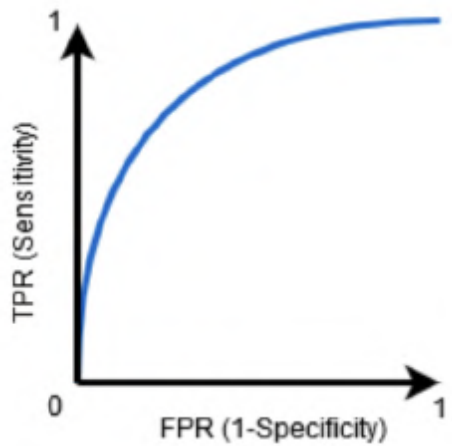
Matrice de confusion

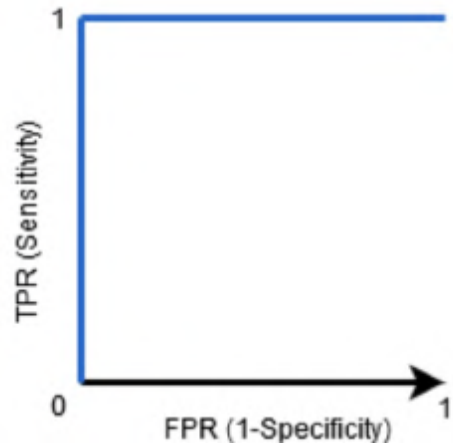
- Un moyen de juger la performance d'un classifier est de faire sa **courbe ROC (ROC curve)**.
- C'est la courbe du "true positive rate / recall" en fonction du "false positive rate" paramétrée par le seuil θ (θ varie de 1 à 0).



Matrice de confusion

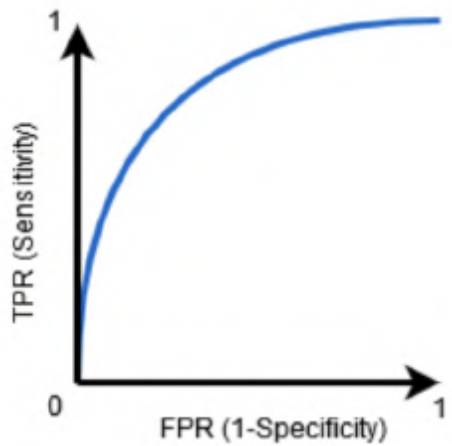
- Pour $\theta = 1$: Tout est classifié "négatif". Ainsi, $FPR = \frac{FP}{N} = 0$ et $TPR = \frac{TP}{P} = 0$. C'est le point $(0, 0)$.
- Pour $\theta = 0$: Tout est classifié "positif". Ainsi, $FPR = \frac{FP}{N} = \frac{N}{N} = 1$ et $TPR = \frac{TP}{P} = \frac{P}{P} = 1$. C'est le point $(1, 1)$.
- Le point $(0, 1)$ représente le classifieur parfait: $FPR = \frac{FP}{N} = 0 \Rightarrow FP = 0$ et $TPR = \frac{TP}{P} = 1 \Rightarrow TP = P$.

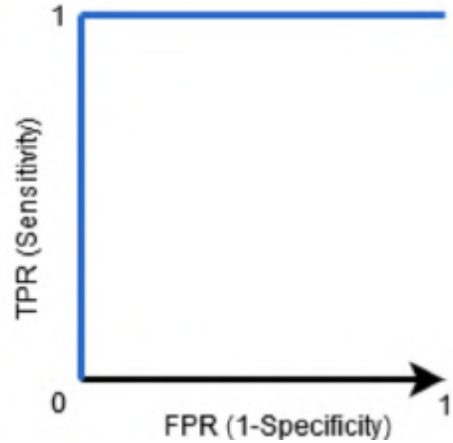




Matrice de confusion

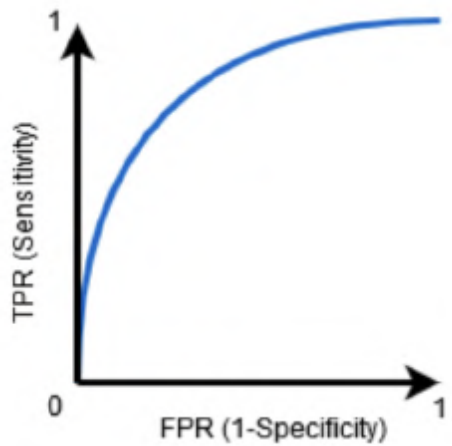
- Pour $\theta = 1$: Tout est classifié "négatif". Ainsi, $FPR = \frac{FP}{N} = 0$ et $TPR = \frac{TP}{P} = 0$. C'est le point $(0, 0)$.
- Pour $\theta = 0$: Tout est classifié "positif". Ainsi, $FPR = \frac{FP}{N} = \frac{N}{N} = 1$ et $TPR = \frac{TP}{P} = \frac{P}{P} = 1$. C'est le point $(1, 1)$.
- Le point $(0, 1)$ représente le classifieur parfait: $FPR = \frac{FP}{N} = 0 \Rightarrow FP = 0$ et $TPR = \frac{TP}{P} = 1 \Rightarrow TP = P$.

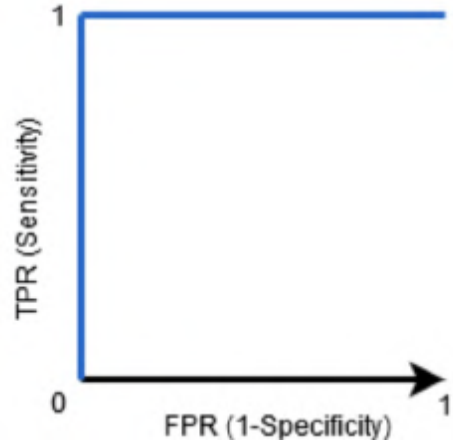




Matrice de confusion

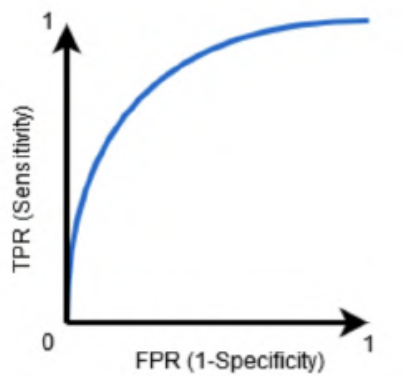
- Pour $\theta = 1$: Tout est classifié "négatif". Ainsi, $FPR = \frac{FP}{N} = 0$ et $TPR = \frac{TP}{P} = 0$. C'est le point $(0, 0)$.
- Pour $\theta = 0$: Tout est classifié "positif". Ainsi, $FPR = \frac{FP}{N} = \frac{N}{N} = 1$ et $TPR = \frac{TP}{P} = \frac{P}{P} = 1$. C'est le point $(1, 1)$.
- Le point $(0, 1)$ représente le classifieur parfait: $FPR = \frac{FP}{N} = 0 \Rightarrow FP = 0$ et $TPR = \frac{TP}{P} = 1 \Rightarrow TP = P$.

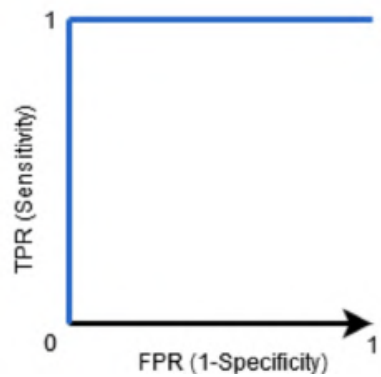




Matrice de confusion

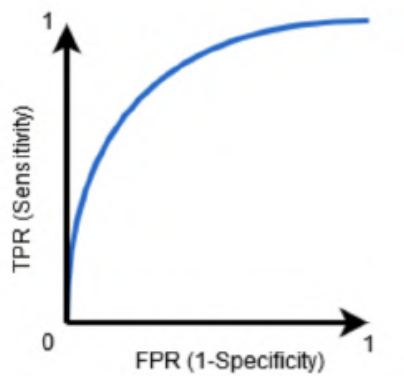
- Ainsi, on peut évaluer la performance d'un classifier en mesurant de combien il s'écarte du classifieur parfait.
- Pour cela, on mesure l'aire sous la courbe ROC, appelée **area under curve (AUC)**.
- Si $AUC = 1$, on est dans le cas du classifieur parfait. Si $AUC = 0.5$, c'est un classifieur random. Si $AUC = 0$, on a un classifieur qui classifie tout à l'envers (donc "anti-parfait").

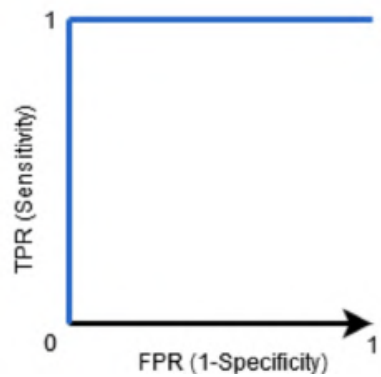




Matrice de confusion

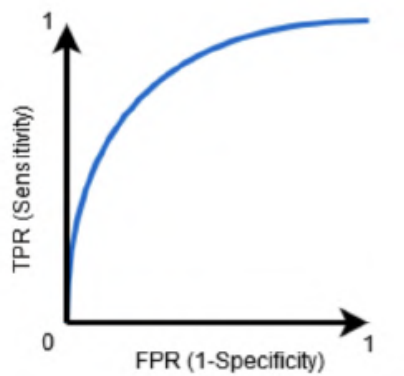
- Ainsi, on peut évaluer la performance d'un classifier en mesurant de combien il s'écarte du classifieur parfait.
- Pour cela, on mesure l'aire sous la courbe ROC, appelée **area under curve (AUC).**
- Si $AUC = 1$, on est dans le cas du classifieur parfait. Si $AUC = 0.5$, c'est un classifieur random. Si $AUC = 0$, on a un classifieur qui classifie tout à l'envers (donc "anti-parfait").

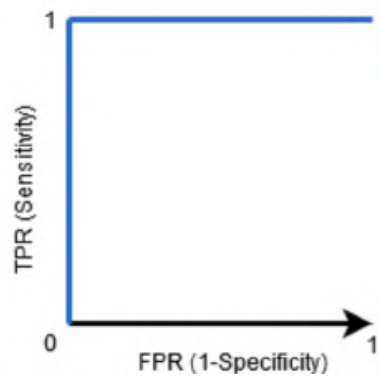




Matrice de confusion

- Ainsi, on peut évaluer la performance d'un classifier en mesurant de combien il s'écarte du classifieur parfait.
- Pour cela, on mesure l'aire sous la courbe ROC, appelée **area under curve (AUC).**
- Si $AUC = 1$, on est dans le cas du classifieur parfait. Si $AUC = 0.5$, c'est un classifieur random. Si $AUC = 0$, on a un classifieur qui classifie tout à l'envers (donc "anti-parfait").





Bibliographie


James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013).
An Introduction to Statistical Learning: with Applications in R, volume 103 of *Springer Texts in Statistics*.
Springer, New York.