ZOIDBERG 2.0

BOOTSTRAP

1. Introduction au Machine Learning
2. Réaliser un projet de machine learning
3. Modélisation
4. Réduction de dimension
5. Les algorithmes de classification
6. Évaluation du modèle
7. Optimisation des hyperparamètres

ZOIDBERG 2.0

BOOTSTRAP

5. Les algorithmes de classification
   - La régression logistique
   - Les arbres de décision
   - Les séparateurs à vastes marge
   - Les K plus proches voisins

RÉGRESSION LOGISTIQUE

C'est un algorithme de **classification**

Le but de la régression logistique est de **donner une** estimation de la probabilité qu'un évènement se produise et de **déterminer une relation** entre des variables explicatives et les probabilités de résultats.

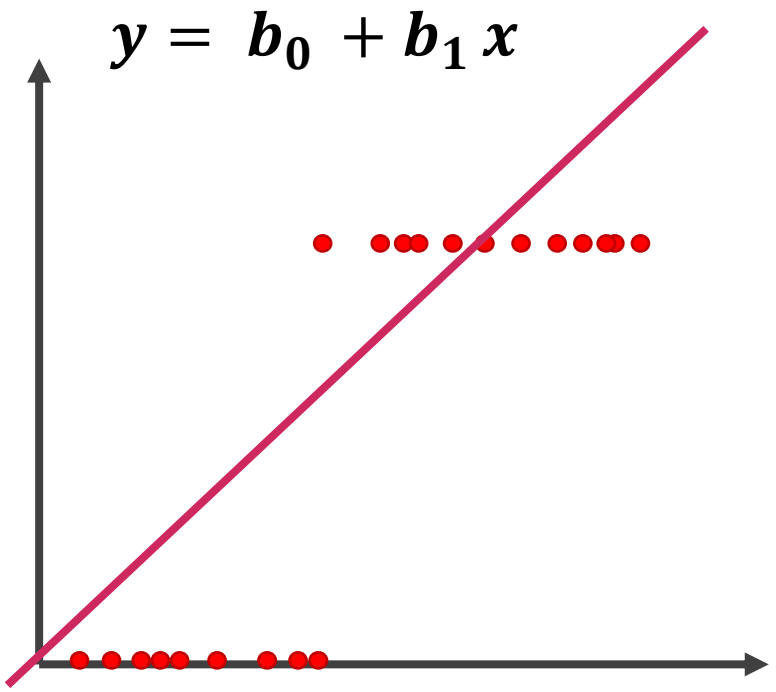
RÉGRESSION LOGISTIQUE

C'est un algorithme de **classification**

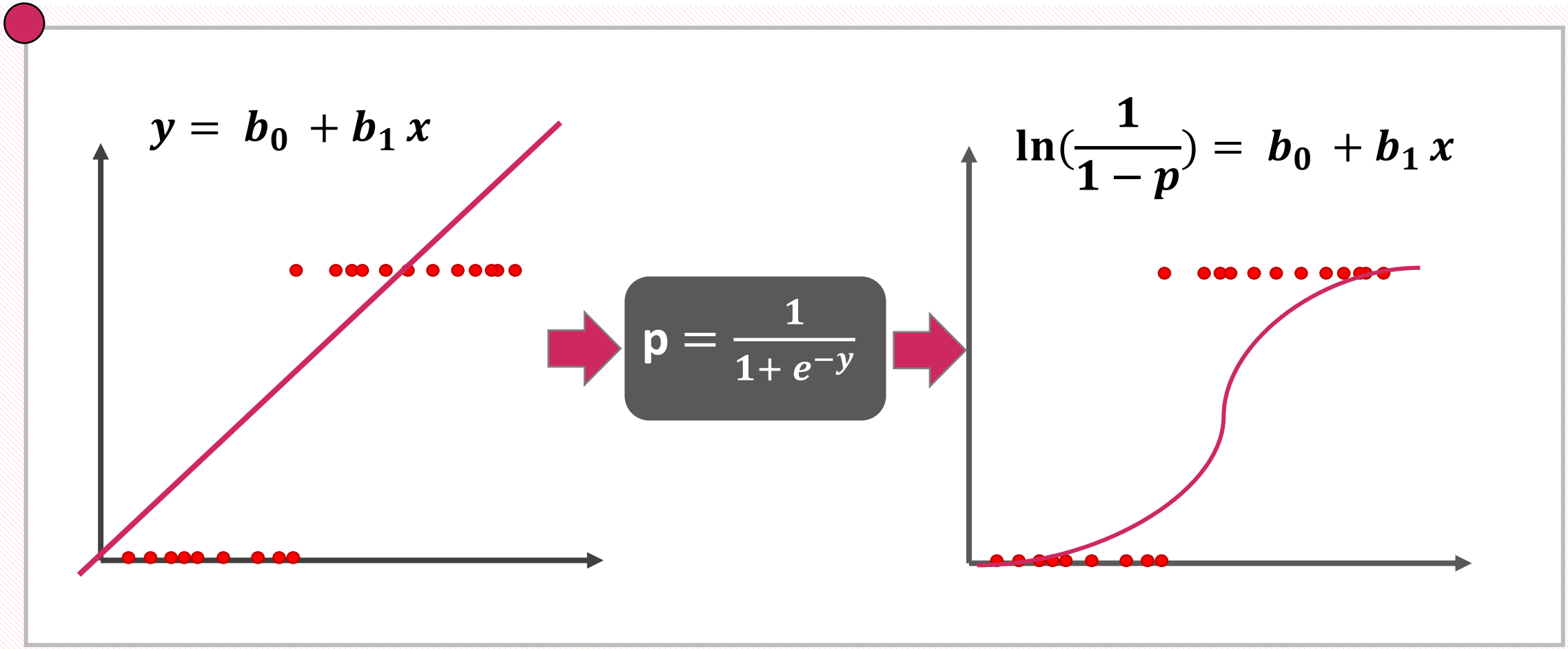
Le but de la régression logistique est de **donner une** estimation de la probabilité qu'un évènement se produise et de **déterminer une relation** entre des variables explicatives et les probabilités de résultats.

- **Régression logistique binaire** ex : oui/non ; malade/sain
- **Régression logistique multinomiale** : la variable dépendante a trois catégories nominales ou plus ex: prédiction de la catégorie d'iris
- **Régression logistique ordinale** : la variable dépendante a trois catégories ordinales ou plus, ex : la notation de produits de 1 à 5

RÉGRESSION LOGISTIQUE

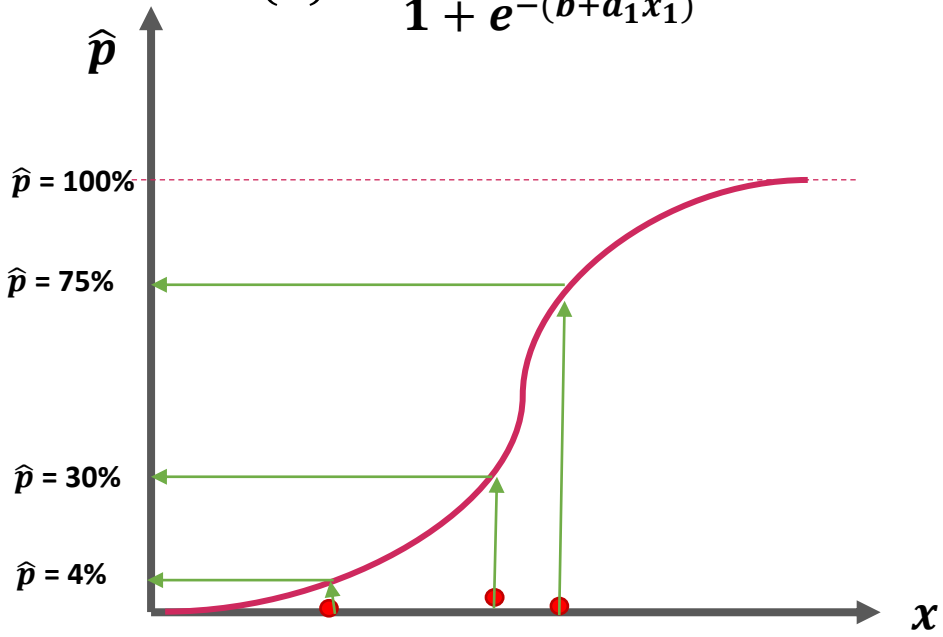


RÉGRESSION LOGISTIQUE

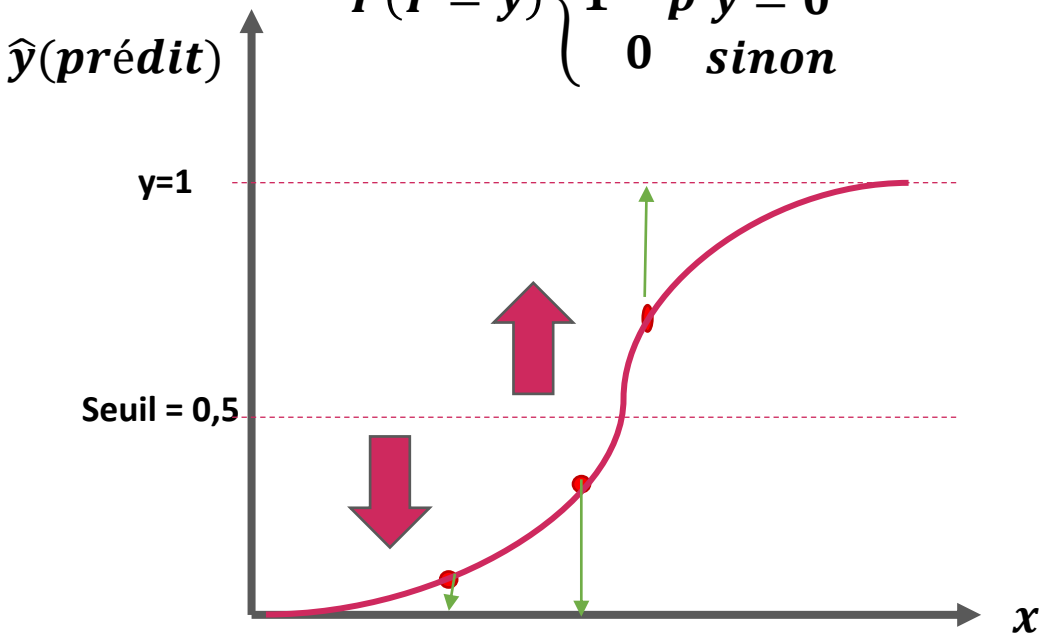


RÉGRESSION LOGISTIQUE

$$P(Y) = \frac{1}{1 + e^{-(b+a_1 x_1)}}$$



$$P(Y = y)\begin{cases} p & y = 1 \\ 1 - p & y = 0 \\ 0 & sinon \end{cases}$$



Y=0

RÉGRESSION LOGISTIQUE

- le seuil que nous définissons correspond à notre critère de classification, généralement il est pris comme valant 0.5
- Lorsque la valeur prédite est supérieure à 0.5, l'événement est susceptible de se produire
- Lorsque la valeur prédite est inférieure à 0.5, il ne l'est pas.

$$P\,(Y = y) \begin{cases} p & y = 1 \\ 1 - p & y = 0 \\ 0 & sinon \end{cases}$$

Y=0

RÉGRESSION LOGISTIQUE

```
#import du module
from sklearn.linear_model import LogisticRegression
# on définit notre estimateur
model = LogisticRegression()
# on on l'applique à nos données :
Model.fit(X_train,  y_train)
# pour obtenir le résultat de la prédiction :
model.predict( )
#pour obtenir la probabilité de la prédiction on utilise la méthode :
Predict_proba( )
# si l'on souhaite changer le seuil :
```

https://stackoverflow.com/questions/31417487/sklearn-logisticregression-and-changing-the-default-threshold-for-classification

ZOIDBERG 2.0 BOOTSTRAP

5. Les algorithmes de classification
   - La régression logistique
   - Les arbres de décision
   - Les séparateurs à vastes marge
   - Les K plus proches voisins

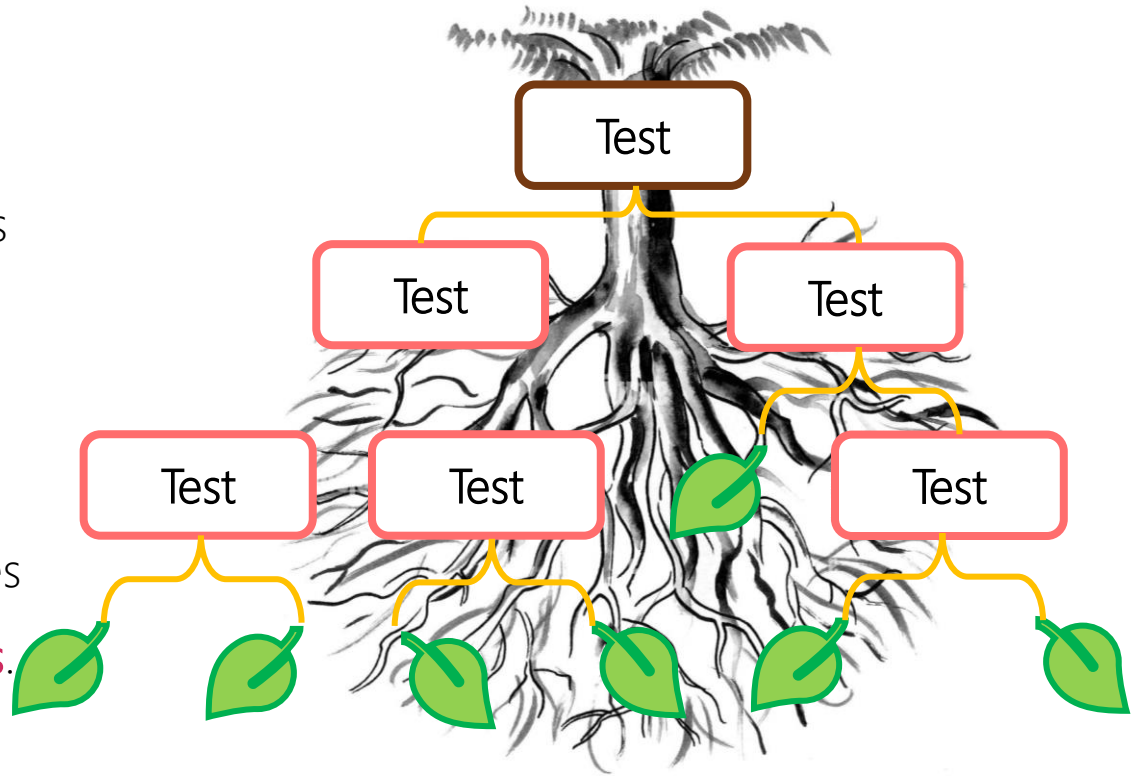
LES ARBRES DE DÉCISIONS

Un arbre de décision est une suite de tests .

Il démarre avec une racine et débouche sur des feuilles.

Chaque test constitue un nœud et établie une règle de décision.

Chaque feuille contient un ensemble de données respectant une ou plusieurs règles de décisions.



LES ARBRES DE DÉCISIONS

Quelles règles de décision choisir pour un cas de classification ?

L'idée est de pouvoir déterminer les tests qui **apporte le plus d'infos** afin de séparer les données de la façon la plus pertinente.

Pour ce faire on s'appuie sur l'un des 2 critères suivants :

- L'entropie : paramètre caractérisant le degré de désorganisation d'un ensemble de donnée. Autrement dit, l'entropie permet de **mesurer le désordre**.
- Le coefficient de Gini : c'est une mesure statistique permettant de rendre compte de la répartition d'une variable au sein d'une population. On l'appelle aussi **coefficient de pureté**.

LES ARBRES DE DÉCISIONS

Par quel test commencer L'arbre de décision ?

Statistiquement le test qui a le plus petit coef de Gini pondéré est celui qui va nous apporter le plus d'informations sur nos données.

$$Gini = 1 - \sum_{classe} \left(\frac{n_{i\ \epsilon\ à\ une\ classe}}{n_{i\ tot}} \right)^2$$

Comment choisir ensuite les nœuds ?

On sélectionne le test ayant le coef de Gini le plus faible de nouveau et on itère jusqu'à que les ensembles de données soient parfaitement purs.

⇨ il s'agit de l'algorithme CART , celui implémenté par défaut dans scikitlearn

LES ARBRES DE DÉCISIONS

patient	malade	Mal de tête ?	Fièvre?	Enrhumé ?
0001	Y	Y	N	Y
0002	N	Y	N	Y
0003	N	Y	Y	N
0004	Y	Y	N	Y
0005	Y	Y	N	Y
0006	Y	Y	N	Y
0007	N	N	Y	Y
0008	N	Y	N	N
0009	Y	Y	N	Y
0010	N	Y	N	N

$$Gini = 1 - \sum_{classe} \left(\frac{n_{i\ \epsilon\ \text{à une classe}}}{n_{i\ tot}} \right)^2$$

LES ARBRES DE DÉCISIONS

patient	malade	Mal de tête ?	Fièvre?	Enrhumé ?
0001	Y	Y	N	Y
0002	N	Y	N	Y
0003	N	Y	Y	N
0004	Y	Y	N	Y
0005	Y	Y	N	Y
0006	Y	Y	N	Y
0007	N	N	Y	Y
0008	N	Y	N	N
0009	Y	Y	N	Y
0010	N	Y	N	N

$$Gini = 1 - \sum_{classe} \left(\frac{n_{i\ \epsilon\ \text{à une classe}}}{n_{i\ tot}} \right)^2$$

$$Gini = 1 - \left(\frac{5}{10}\right)^2 - \left(\frac{5}{10}\right)^2 = 0.5$$

Nous avons une chance sur 2 d'avoir une personne malade dans notre dataset.

LES ARBRES DE DÉCISIONS

patient	malade	Mal de tête ?	Fièvre?	Enrhumé ?
0001	Y	Y	N	Y
0002	N	Y	N	Y
0003	N	Y	Y	N
0004	Y	Y	N	Y
0005	Y	Y	N	Y
0006	Y	Y	N	Y
0007	N	N	Y	Y
0008	N	Y	N	N
0009	Y	Y	N	Y
0010	N	Y	N	N

$$Par\ quel\ test\ commencer\ ?$$

Mal de tête ?
Fièvre ?
Enrhumé ?

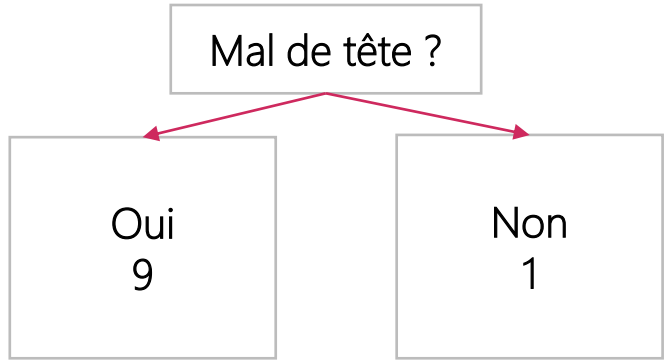
on calcule alors le coef de Gini pondéré pour chaque test

$$\boldsymbol{Gini_{test}} = \sum \left(\frac{n_{i\ \epsilon\ \text{à}\ une\ classe}}{n_{i\ tot}} \right) \times \boldsymbol{Gini_{classe}}$$

$$\boldsymbol{Avec\ Gini_{classe}} = \mathbf{1} - \left(\frac{\boldsymbol{n_{pos}}}{\boldsymbol{n_{tot/class}}} \right)^{\mathbf{2}} - \left(\frac{\boldsymbol{n_{neg}}}{\boldsymbol{n_{tot/class}}} \right)^{\mathbf{2}}$$

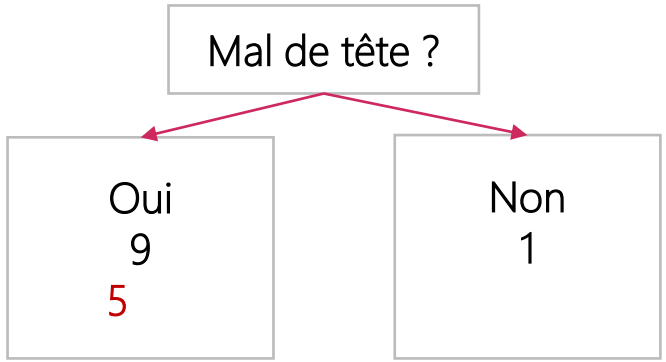
LES ARBRES DE DÉCISIONS

patient	malade	Mal de tête ?	Fièvre?	Enrhumé ?
0001	Y	Y	N	Y
0002	N	Y	N	Y
0003	N	Y	Y	N
0004	Y	Y	N	Y
0005	Y	Y	N	Y
0006	Y	Y	N	Y
0007	N	N	Y	Y
0008	N	Y	N	N
0009	Y	Y	N	Y
0010	N	Y	N	N



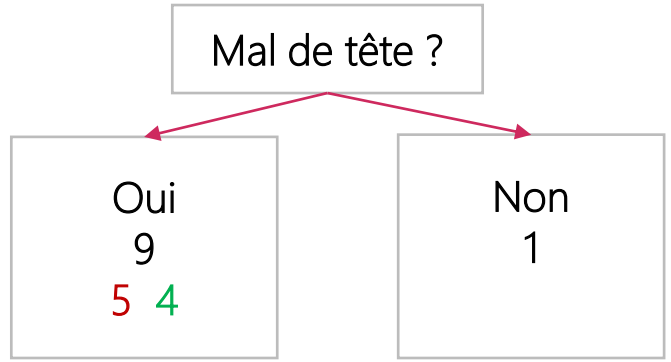
LES ARBRES DE DÉCISIONS

patient	malade	Mal de tête ?	Fièvre?	Enrhumé ?
0001	Y	Y	N	Y
0002	N	Y	N	Y
0003	N	Y	Y	N
0004	Y	Y	N	Y
0005	Y	Y	N	Y
0006	Y	Y	N	Y
0007	N	N	Y	Y
0008	N	Y	N	N
0009	Y	Y	N	Y
0010	N	Y	N	N



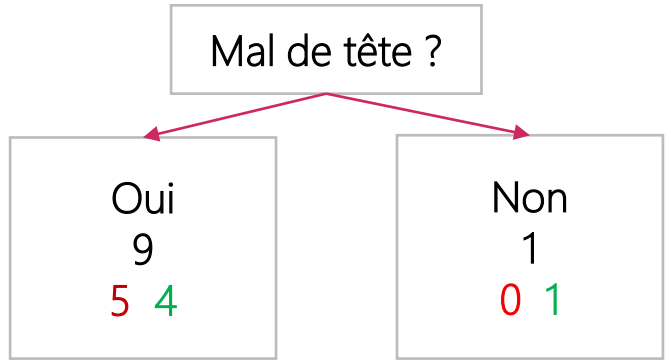
LES ARBRES DE DÉCISIONS

patient	malade	Mal de tête ?	Fièvre?	Enrhumé ?
0001	Y	Y	N	Y
0002	N	Y	N	Y
0003	N	Y	Y	N
0004	Y	Y	N	Y
0005	Y	Y	N	Y
0006	Y	Y	N	Y
0007	N	N	Y	Y
0008	N	Y	N	N
0009	Y	Y	N	Y
0010	N	Y	N	N



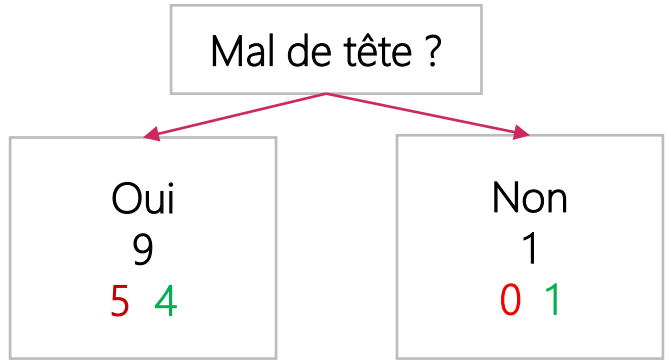
LES ARBRES DE DÉCISIONS

patient	malade	Mal de tête ?	Fièvre?	Enrhumé ?
0001	Y	Y	N	Y
0002	N	Y	N	Y
0003	N	Y	Y	N
0004	Y	Y	N	Y
0005	Y	Y	N	Y
0006	Y	Y	N	Y
0007	N	N	Y	Y
0008	N	Y	N	N
0009	Y	Y	N	Y
0010	N	Y	N	N



LES ARBRES DE DÉCISIONS

patient	malade	Mal de tête ?	Fièvre?	Enrhumé ?
0001	Y	Y	N	Y
0002	N	Y	N	Y
0003	N	Y	Y	N
0004	Y	Y	N	Y
0005	Y	Y	N	Y
0006	Y	Y	N	Y
0007	N	N	Y	Y
0008	N	Y	N	N
0009	Y	Y	N	Y
0010	N	Y	N	N



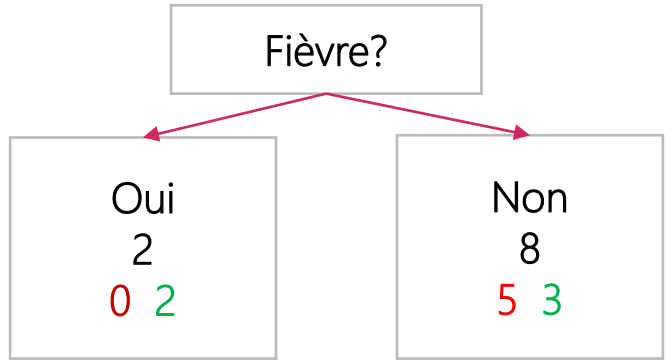
$$G_{malade} = 1 - \left(\frac{5}{9}\right)^2 - \left(\frac{4}{9}\right)^2 = 0.49$$

$$G_{sain} = 1 - \left(\frac{0}{1}\right)^2 - \left(\frac{1}{1}\right)^2 = 0$$

$$G_{tête} = \frac{9}{10} \times 0.49 + 0 = 0.44$$

LES ARBRES DE DÉCISIONS

patient	malade	Mal de tête ?	Fièvre?	Enrhumé ?
0001	Y	Y	N	Y
0002	N	Y	N	Y
0003	N	Y	Y	N
0004	Y	Y	N	Y
0005	Y	Y	N	Y
0006	Y	Y	N	Y
0007	N	N	Y	Y
0008	N	Y	N	N
0009	Y	Y	N	Y
0010	N	Y	N	N



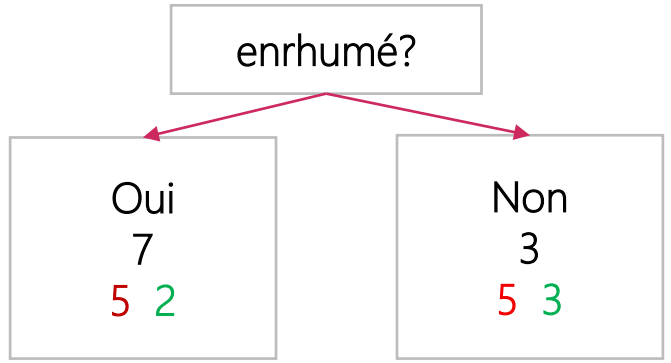
$$G_{malade} = 1 - \left(\frac{2}{2}\right)^2 - \left(\frac{0}{2}\right)^2 = 0$$

$$G_{sain} = 1 - \left(\frac{3}{8}\right)^2 - \left(\frac{5}{8}\right)^2 = 0.46$$

$$G_{fièvre} = \frac{8}{10} \times 0.46 + 0 = 0.37$$

LES ARBRES DE DÉCISIONS

patient	malade	Mal de tête ?	Fièvre?	Enrhumé ?
0001	Y	Y	N	Y
0002	N	Y	N	Y
0003	N	Y	Y	N
0004	Y	Y	N	Y
0005	Y	Y	N	Y
0006	Y	Y	N	Y
0007	N	N	Y	Y
0008	N	Y	N	N
0009	Y	Y	N	Y
0010	N	Y	N	N



$$G_{malade} = 1 - \left(\frac{2}{7}\right)^2 - \left(\frac{5}{7}\right)^2 = 0.40$$

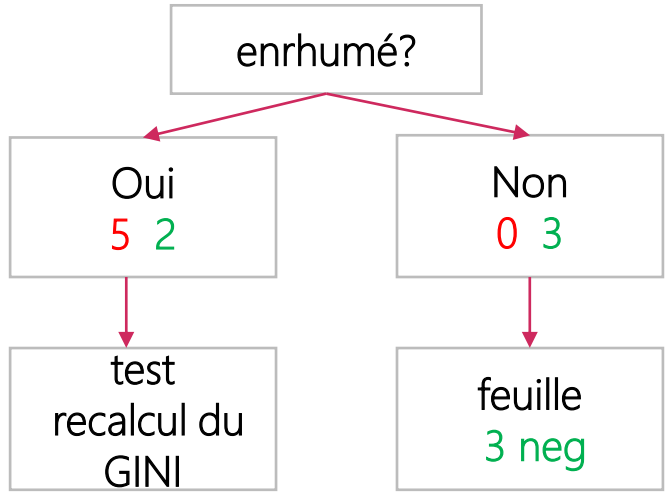
$$G_{sain} = 1 - \left(\frac{3}{3}\right)^2 - \left(\frac{0}{3}\right)^2 = 0$$

$$G_{enrhumé} = \frac{7}{10} \times 0.40 + 0 = 0.27$$

LES ARBRES DE DÉCISIONS

patient	malade	Mal de tête ?	Fièvre?	Enrhumé ?
0001	Y	Y	N	Y
0002	N	Y	N	Y
0003	N	Y	Y	N
0004	Y	Y	N	Y
0005	Y	Y	N	Y
0006	Y	Y	N	Y
0007	N	N	Y	Y
0008	N	Y	N	N
0009	Y	Y	N	Y
0010	N	Y	N	N

$G_{toux} = 0.44$ $G_{fièvre} = 0.37$ $G_{enrhumé} = 0.27$



LES ARBRES DE DÉCISIONS

Attention le problème majeur des arbres de décision est de rentrer en underfitting : si nous avons trop de feuilles, l'arbre devient trop fidèle aux données d'entraînement et ne pourra être généralisable.

C'est pourquoi il est possible d'appliquer une phase d'élagage pour éviter un trop pleins de feuilles. Pour se faire nous pouvons jouer sur les hyperparamètres suivants:

- max_depht : c'est ce qu'on appelle la régularisation : on régule le split afin d'éviter qu'il soit trop précis . Cela a pour conséquence de diminuer en performance .
- max_feature : le nombre de features prises en considération pour chaque split .

LES ARBRES DE DÉCISIONS

```
#import du module
from sklearn.linear_model import tree # pour tous les cas
from sklearn.tree import DecisionTreeRegressor # pour les cas de regression
from sklearn.tree import DecisionTreeClassifier # pour les cas de classification
# on définit notre estimateur
model = DecisionTreeRegressor( hyperparamètres)
# et on l'applique à nos données :
Model.fit(X_train,  y_train)
#quelques méthodes utiles :
get.depht( ) # retourne la profondeur ou le nombre de nœuds de l'arbre
get.n_leaves( ) # retourne le nombre de feuilles obtenues
# voir la doc : https://scikit-learn.org/stable/modules/tree.html
```

LES ARBRES DE DÉCISIONS

Avantages

- Ils sont faciles à interpréter et à comprendre. Les règles de décisions sont connues
- Ils nécessitent très peu de préparation dans les données . Il n'est pas nécessaire de normaliser les données par exemple.
- Peut gérer des problèmes multi-classes
- Les arbres de décisions font partie des algorithmes les plus puissants et performants

Inconvénients

- risque d'overfitting, les arbres peuvent être trop complexes : il faut donc faire attention à sa profondeur. Plus il y a de nœuds, plus il y a de règles.
- Ils peuvent être instables: de petites variations dans les données peuvent entraîner la génération d'un arbre complètement différent.
- Ils peuvent être biaisés si certaines classes dominent. Il est donc recommandé d'équilibrer le dataset au préalable
- Nous n'avons aucune garantie d'obtenir un arbre optimal

ZOIDBERG 2.0

BOOTSTRAP

5. Les algorithmes de classification
   - La régression logistique
   - Les arbres de décision
   - Les séparateurs à vastes marge
   - Les K plus proches voisins

SÉPARATEUR À VASTE MARGE

- Le SVM est une Machine à vecteur de support ou séparateur à vaste marge.
- C'est un algorithme d'apprentissage supervisé qui obtient de très bonnes performances , du même ordre que celles d'un réseau de neurones.
- Il est utilisé dans un cadre binaire (0-1) pour la classification. Dans un cas de régression ou utilisera le SVR ou support vector regression;
- Ils peuvent travailler avec des données de grandes dimensions et ont un faible nombre d'hyperparamètres.

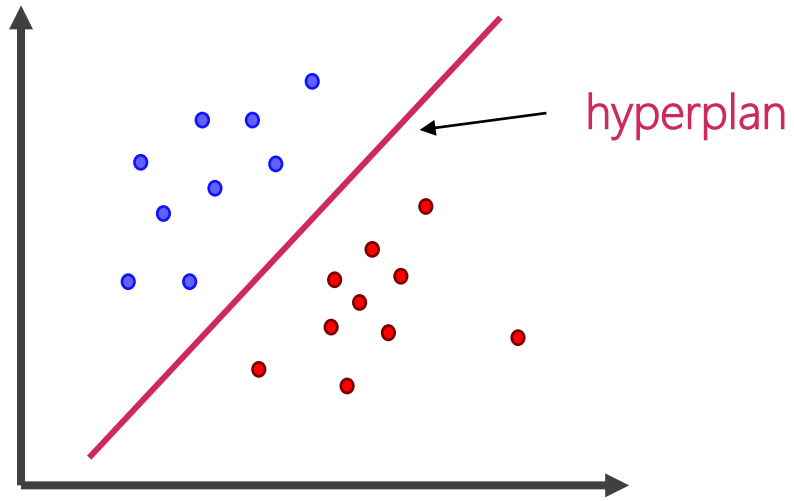
SÉPARATEUR À VASTE MARGE

NOTION D'HYPERPLAN

L'objectif est de trouver une séparation linéaire séparant les 2 classes :

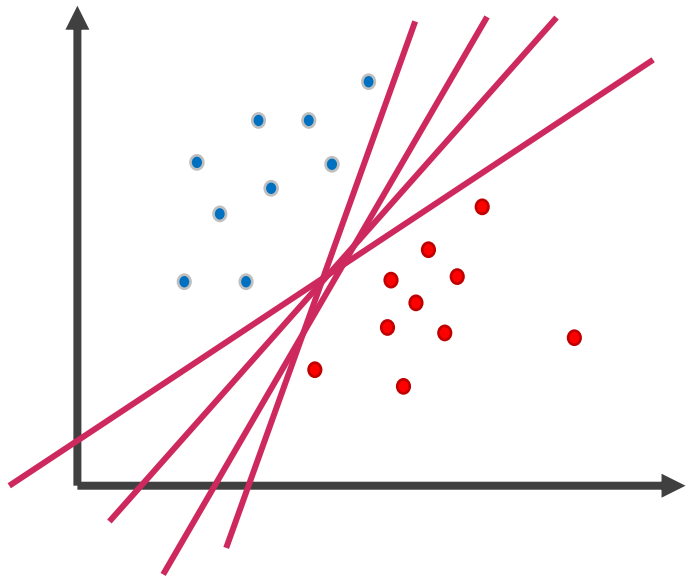
c'est ce que l'on appelle un hyperplan

C'est un plan ou d'un espace de décision qui est divisé entre un ensemble d'objets de classes différentes.



SÉPARATEUR À VASTE MARGE

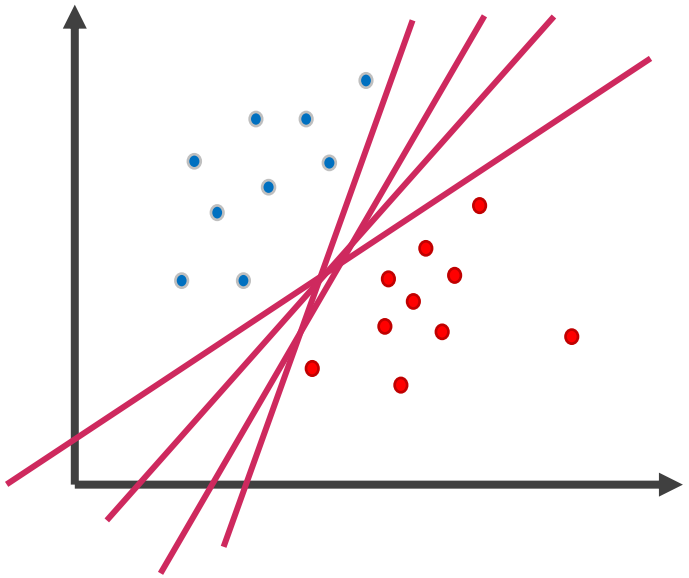
Problème : Plusieurs hyperplans sont valides pour diviser l'espace en deux !



SÉPARATEUR À VASTE MARGE

Le SVM propose l'hyperplan optimal : **celui qui passe « au milieu » des points des deux classes**

De sorte à maximiser la distance entre ces deux classes.

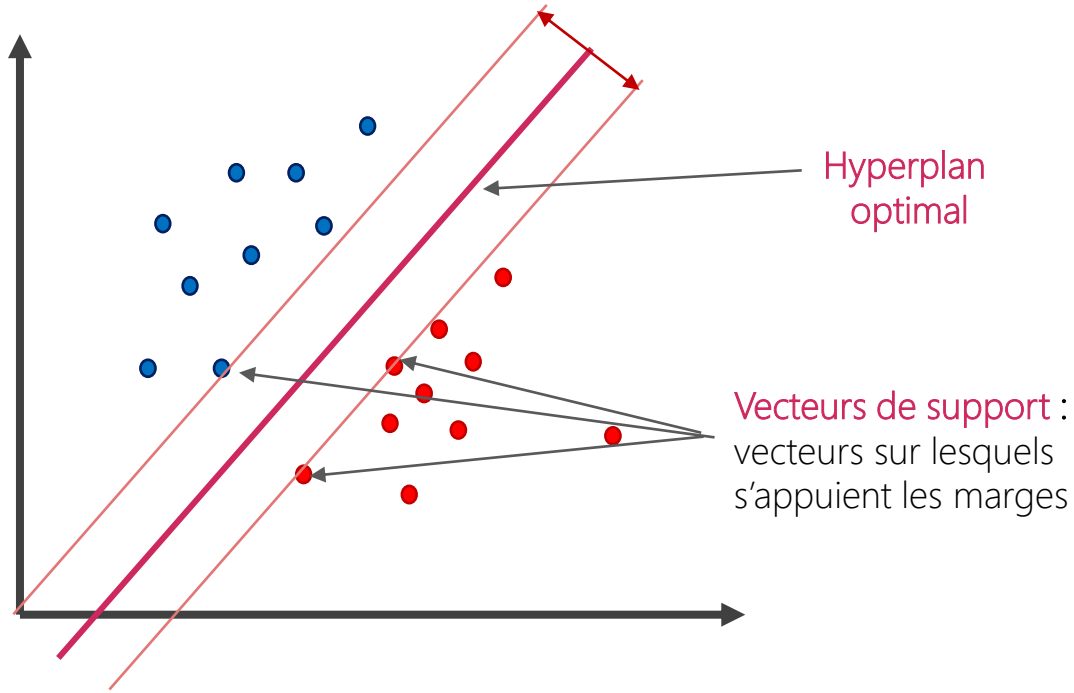


plus les points sont loin de l'hyperplan, plus on est sûr du classement.

Le but → que nos points soient aussi éloignés que possible de l'hyperplan, tout en restant du bon côté.

SÉPARATEUR À VASTE MARGE

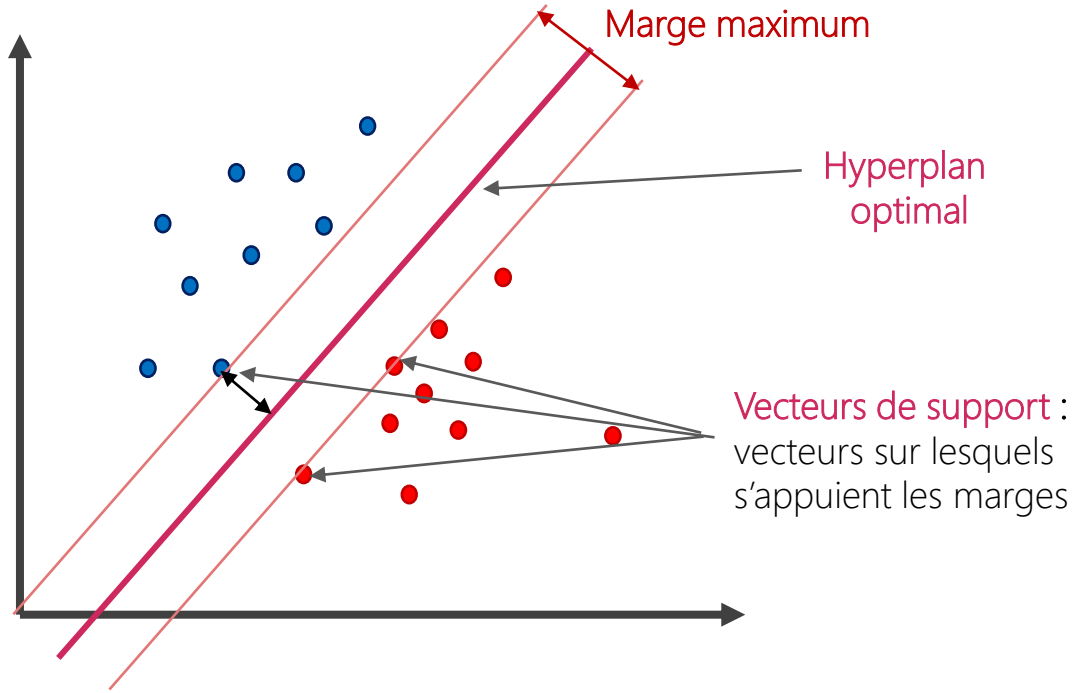
Comment trouver le bon hyperplan?



Les points de données les plus proches de l'hyperplan sont appelés vecteurs de support. La ligne de séparation ou hyperplan optimal sera définie à l'aide de ces points de données.

SÉPARATEUR À VASTE MARGE

Comment trouver le bon hyperplan?



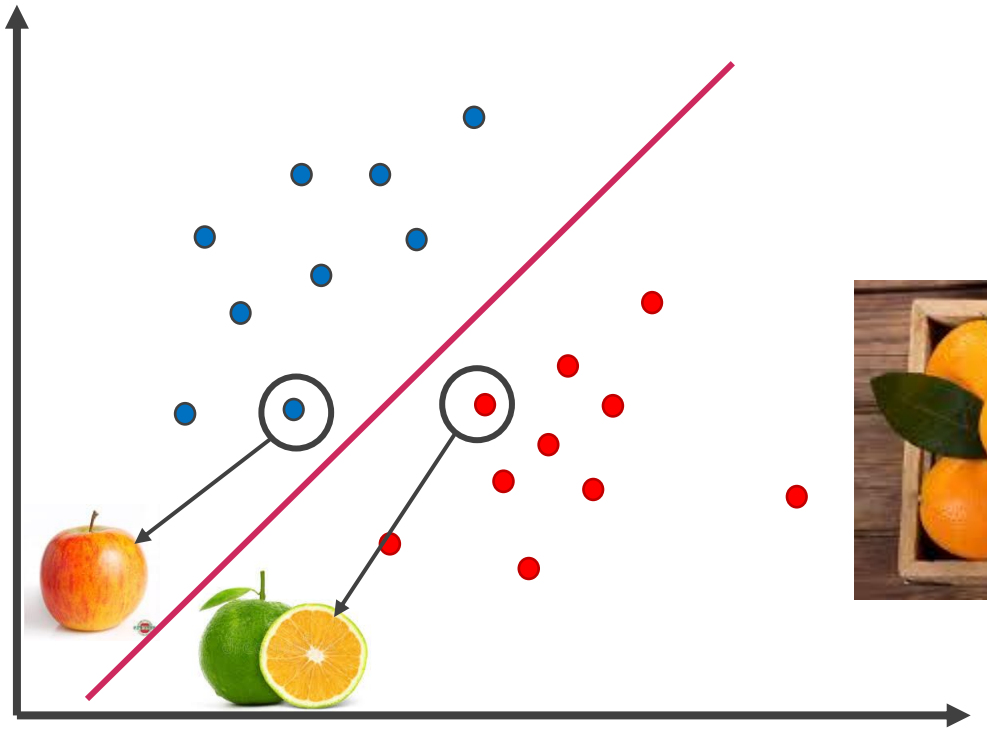
La distance entre l'hyperplan et le point le plus proche de l'un des ensembles est appelée la marge.

On choisira un hyperplan avec la plus grande marge possible entre l'hyperplan et n'importe quel point de l'entraînement : C'est ce qu'on appelle la marge maximale.

On augmente ainsi les chances que de nouvelles données soient classées correctement.

SÉPARATEUR À VASTE MARGE

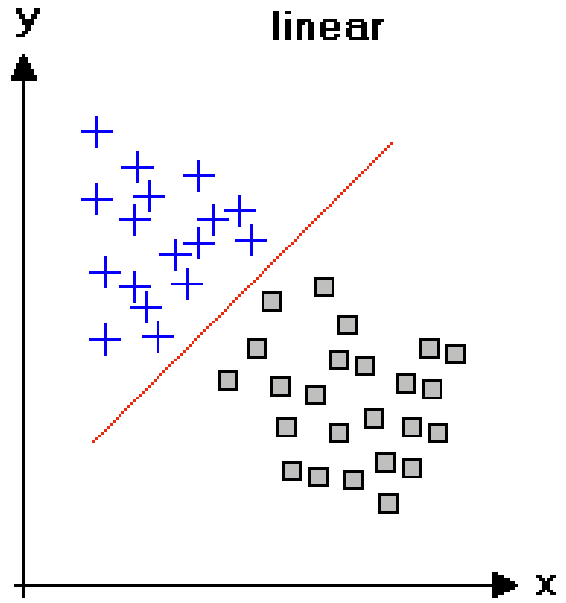
UN EXEMPLE

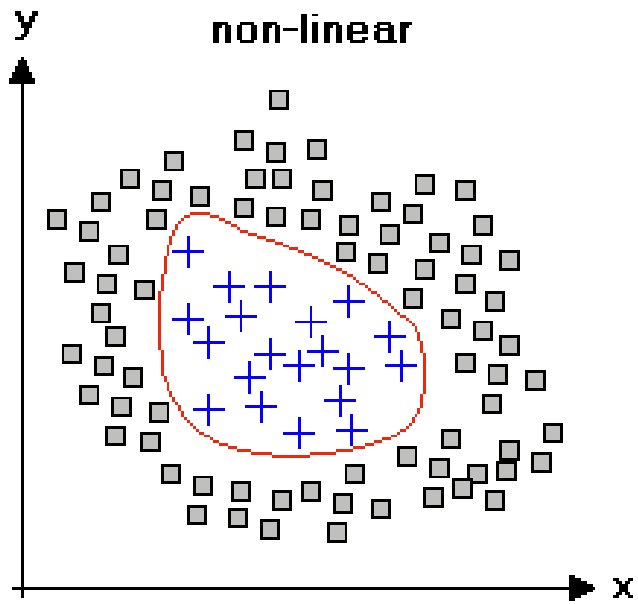


SÉPARATEUR À VASTE MARGE

Que faire si nous ne pouvons pas déterminer d'hyperplan?

Les données ne sont pas toujours linéairement séparables

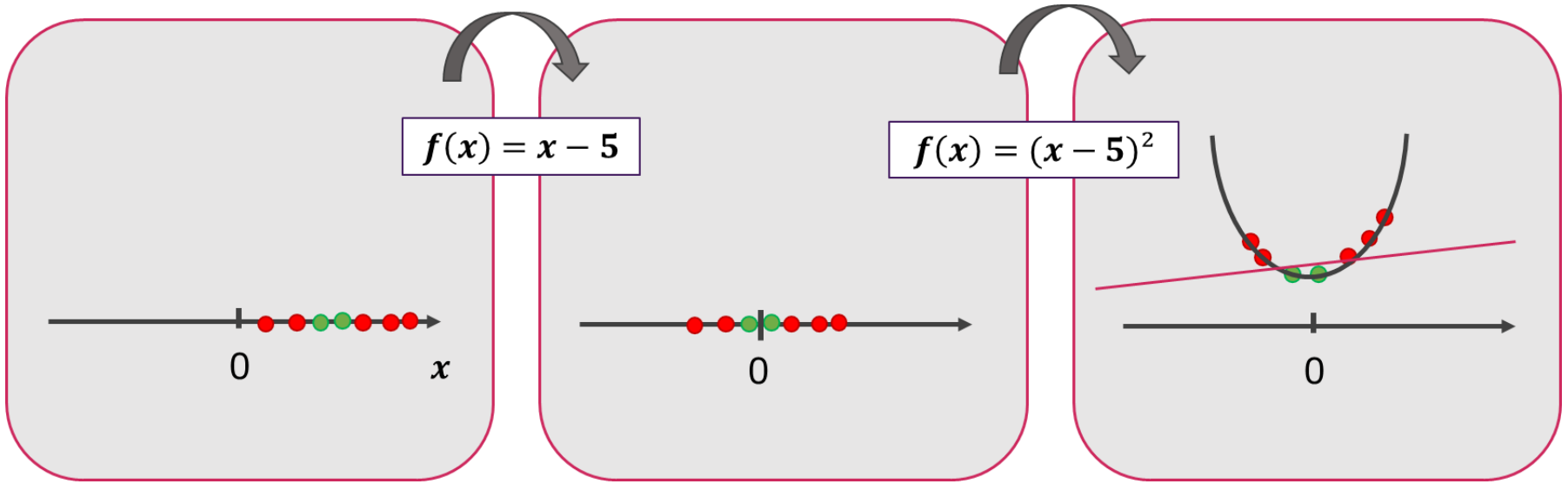




SÉPARATEUR À VASTE MARGE

Solution → on transforme l'espace de représentation des données d'entrées en un espace de plus grande dimension dans lequel il est probable qu'il existe une séparation linéaire

Exemple :



SÉPARATEUR À VASTE MARGE

Problème → atteindre un espace a plus grande dimension peut entrainer une augmentation du temps de calcul ?

On utilise une technique appelée kernel trick ou astuce du noyau qui utilise fonction appelée fonction noyau. Sa particularité est de permettre de transformer un produit scalaire dans un espace de grande dimension en un simple calcul ponctuel de la fonction.

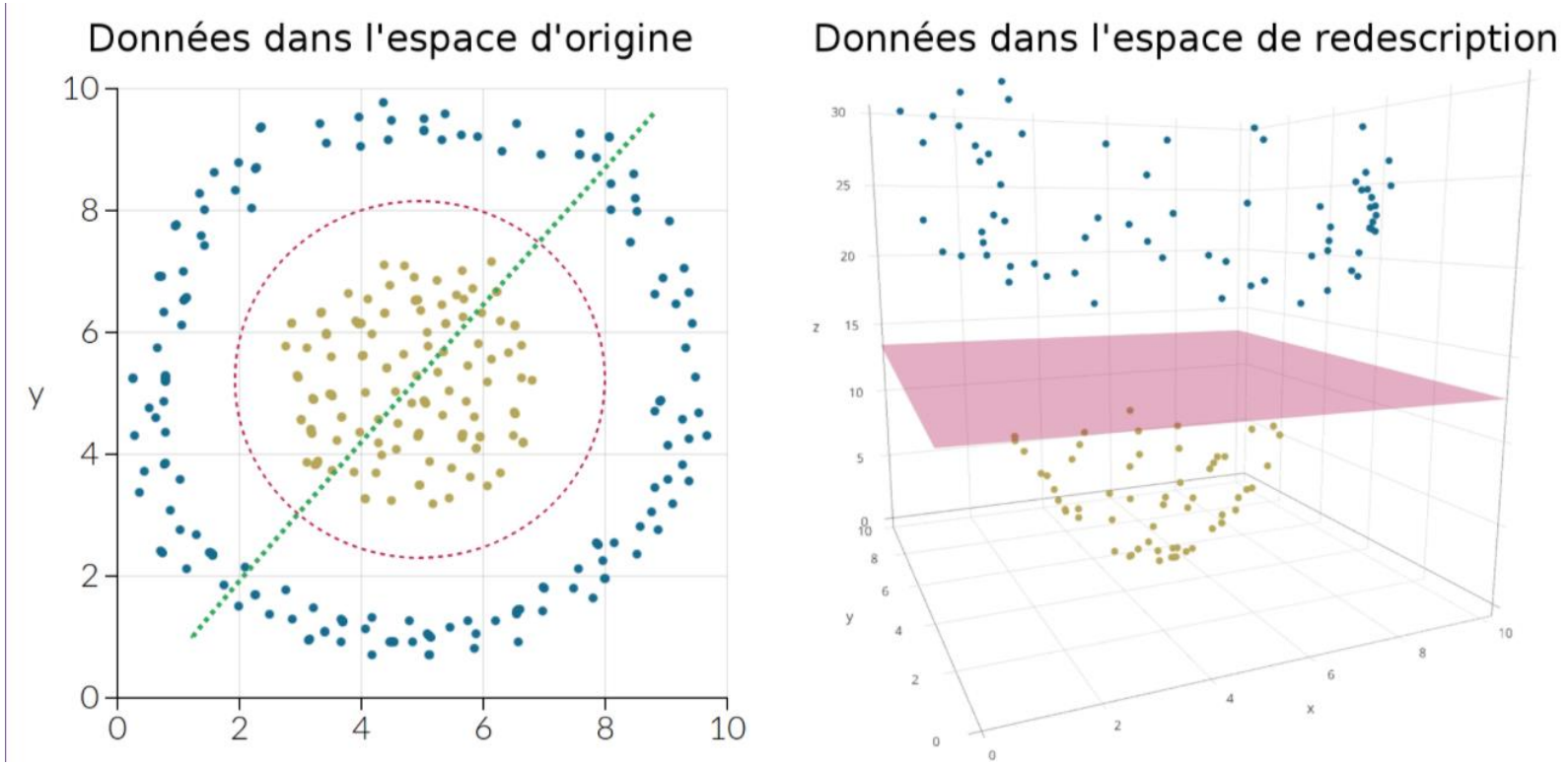
Il n'est donc pas nécessaire de connaitre le calcul explicite de la transformation à appliquer à nos données pour le changement d'espace.

Pour en savoir plus :

- https://www.quora.com/What-are-kernels-in-machine-learning-and-SVM-and-why-do-we-need-them/answer/Lili-Jiang?srid=oOgT
- Kernel Functions. Lately, I have been doing some reading… | by Tejumade Afonja | Towards Data Science

SÉPARATEUR À VASTE MARGE

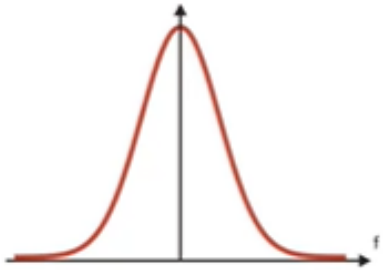
PROBLÈME NON LINÉAIRE : EXEMPLE



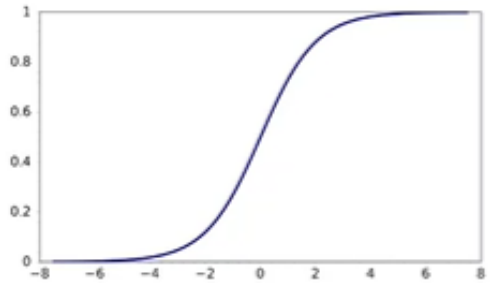
SÉPARATEUR À VASTE MARGE

PROBLÈME NON LINÉAIRE : LES DIFFÉRENTS TYPES DE KERNEL

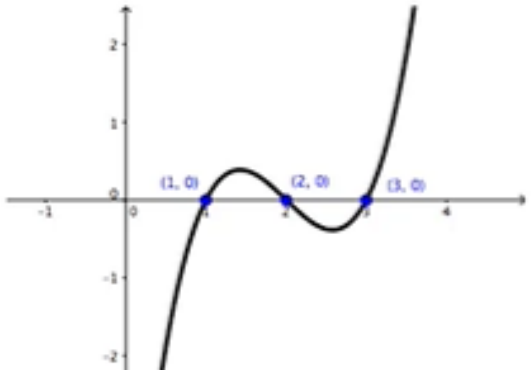
Gaussian RBF kernel



Sigmoid kernel



Polynomial kernel



Pour en savoir plus : https://scikit-learn.org/stable/modules/svm.html#svm-kernels

SÉPARATEUR À VASTE MARGE

LES ESTIMATEURS

`svm.LinearSVC`([penalty, loss, dual, tol, C, ...])	Linear Support Vector Classification.
`svm.LinearSVR`(*[, epsilon, tol, C, loss, ...])	Linear Support Vector Regression.
`svm.NuSVC`(*[, nu, kernel, degree, gamma, ...])	Nu-Support Vector Classification.
`svm.NuSVR`(*[, nu, C, kernel, degree, gamma, ...])	Nu Support Vector Regression.
`svm.OneClassSVM`(*[, kernel, degree, gamma, ...])	Unsupervised Outlier Detection.
`svm.SVC`(*[, C, kernel, degree, gamma, ...])	C-Support Vector Classification.
`svm.SVR`(*[, kernel, degree, gamma, coef0, ...])	Epsilon-Support Vector Regression.
`svm.l1_min_c`(X, y, *[, loss, fit_intercept, ...])	Return the lowest bound for C such that for C in (l1_min_C, infin empty.

Pour la classification

SÉPARATEUR À VASTE MARGE

#les hyperparamètres pour les cas de classification :

- C : Paramètre de régularisation. L'importance de la régularisation est inversement proportionnelle à la valeur de C.
- Kernel : Spécifie le type de kernel utilisé dans l'algorithme.
- Degree : Degré du kernel polynomial ('poly'). A ignorer pour les autres kernels
- Gamma : Coefficient du kernel pour 'rbf', 'poly' and 'sigmoid'.
- Coef0 : terme independent de la fonction du kernel qui n'a de sens que pour les kernel 'poly' et 'sigmoid'.

#les hyperparamètres pour les cas de régression :

- C : Paramètre de régularisation.
- Epsilon : le parameter qui apparaît dans la function de coût
- Loss : determine la fonction de coût utilisée : l1 (MAE) ou l2 (MSE)

SÉPARATEUR À VASTE MARGE

Comment choisir les valeurs des paramètres d'optimisation ?

- Une simulation pour voir l'effet du paramètre C : https://remi.flamary.com/demos/svmreg.fr.html
- Quelques articles intéressants :
  - https://datascience.stackexchange.com/questions/4943/intuition-for-the-regularization-parameter-in-svm
  - https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72

SÉPARATEUR À VASTE MARGE

Avantages

- Ils peuvent traiter des problèmes à dimensions élevées.
- Et restent tout de même efficaces même si le nombre de dimensions est plus élevés que le nombre de données d'entraînement
- Ce sont des algorithmes puissants qui fonctionnent bien sur de petits datasets
- Le paramètre de régularisation permet d 'éviter l'overfitting

Inconvénients

- Les SVM ne fournissent pas directement d'estimations de probabilité, Celles-ci peuvent être obtenues avec une validation croisée coûteuse.
- Ils ne fonctionnent pas très bien dans les cas où les classes se chevauchent
- Ils deviennent couteux en termes de calcul pour de grands datasets et sont sensibles aux outliers.
- Les données doivent être standardisées

Pour en savoir plus sur les SVM 1.4. Support Vector Machines — scikit-learn 1.1.1 documentation

ZOIDBERG 2.0

BOOTSTRAP

5. Les algorithmes de classification
   - La régression logistique
   - Les arbres de décision
   - Les séparateurs à vastes marge
   - Les K plus proches voisins

LES K PLUS PROCHES VOISINS

- C'est un algorithme utilisé aussi bien en machine learning supervisé que non supervisé .

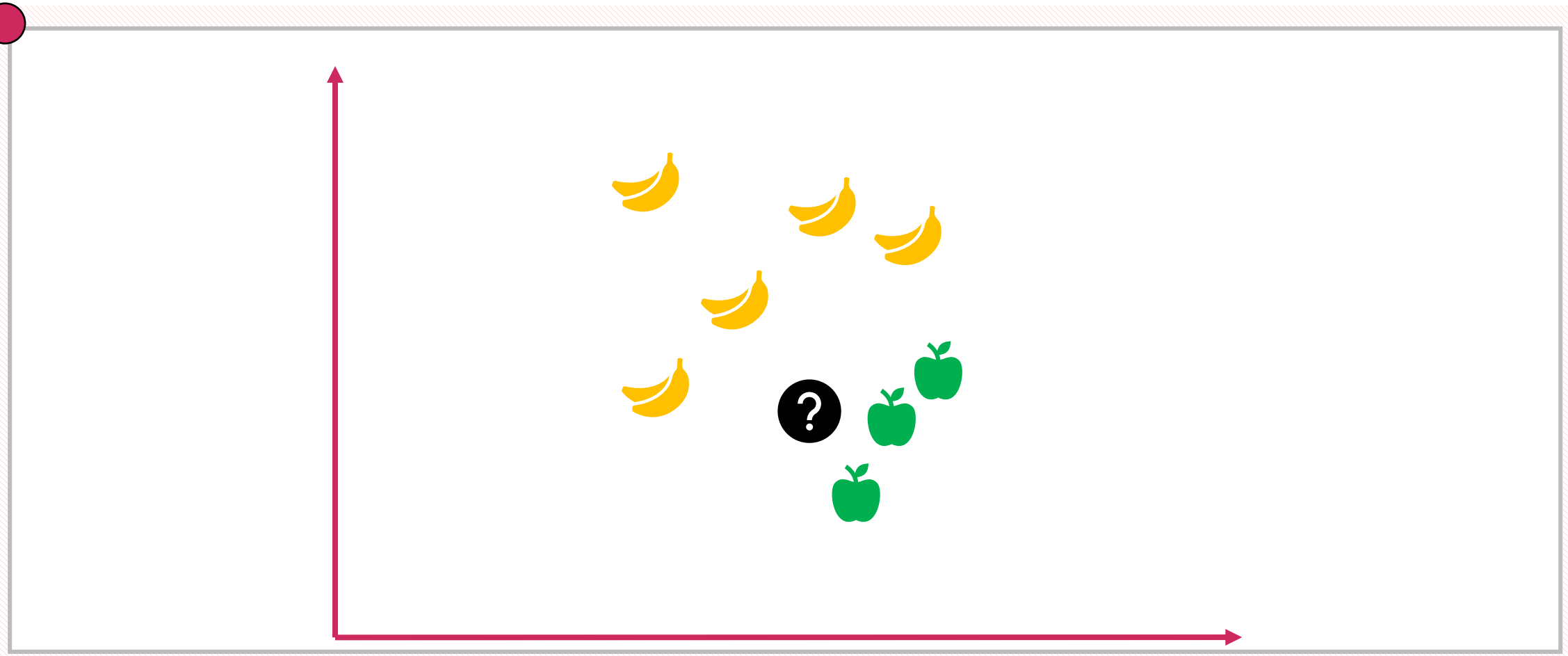
- C'est un algorithme, qui ne construit pas de modèle prédictif . Il n'y a pas vraiment de phase d'apprentissage. Pour effectuer une prédiction, l'algo se base sur le jeu de donnée en entier. La classification est calculée à partir d'un vote à la majorité des voisins les plus proches de chaque point. Autrement dit, on attribue à un nouveau point la classe qui a le plus de représentants parmi les voisins les plus proches de ce point .

LES K PLUS PROCHES VOISINS

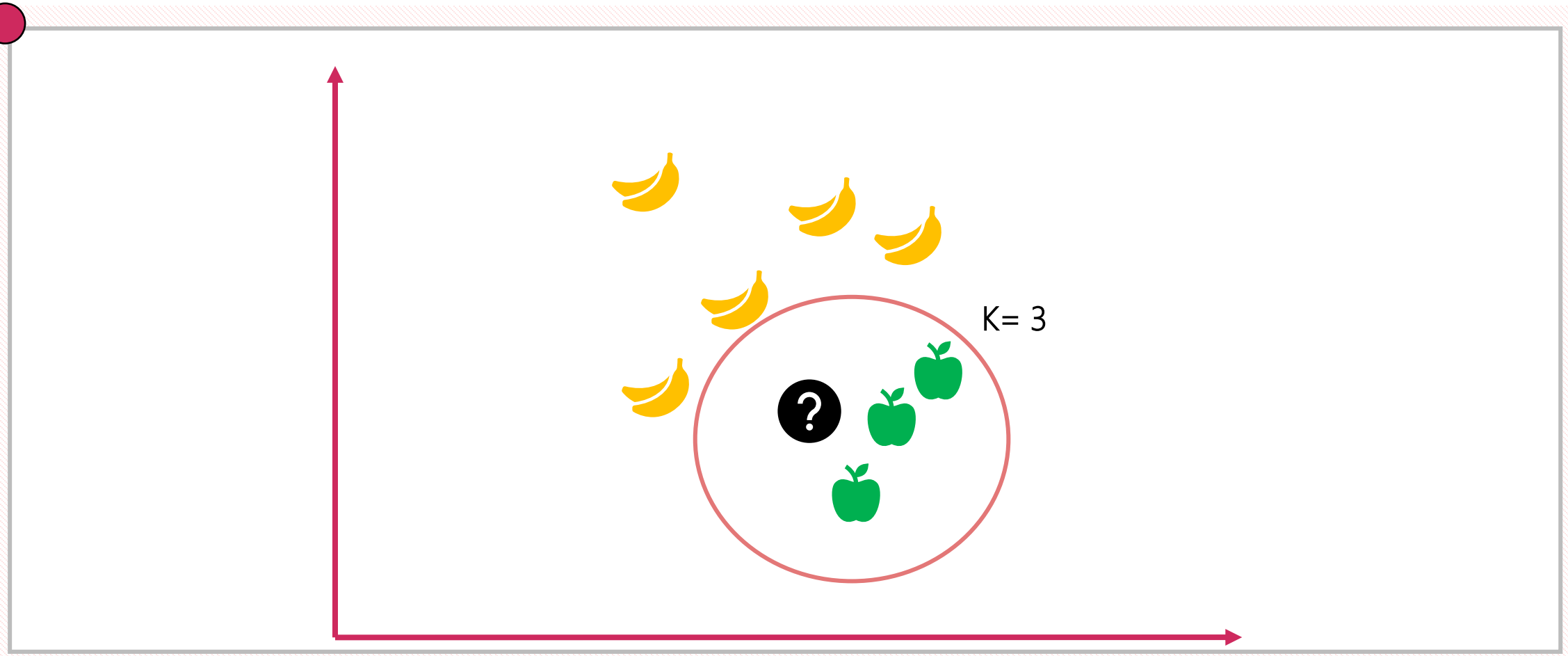
En apprentissage supervisé, on peut l'utilisé pour les problèmes de régression comme de classification .

- Si K-NN est utilisé pour la régression, c'est la moyenne (ou la médiane) des variables y des K plus proches observations qui servira pour la prédiction.
- Si K-NN est utilisé pour la classification, c'est le mode des variables y des K plus proches observations qui servira pour la prédiction

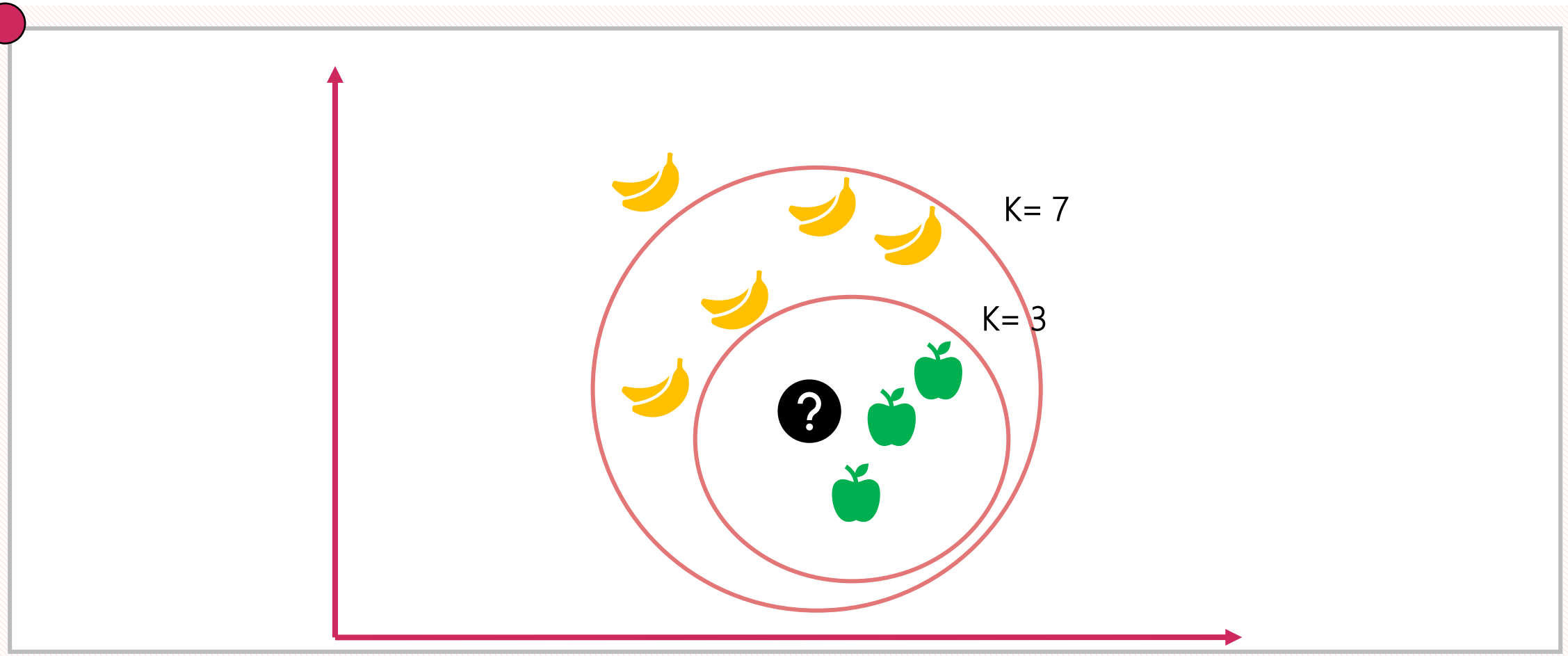
LES K PLUS PROCHES VOISINS



LES K PLUS PROCHES VOISINS



LES K PLUS PROCHES VOISINS



LES K PLUS PROCHES VOISINS

Étape 1

- On choisit le nombre de K voisins les plus proches que l'on souhaite prendre en considération

LES K PLUS PROCHES VOISINS

Étape 1

- On choisit le nombre de K voisins les plus proches que l'on souhaite prendre en considération

Étape 2

- On calcule la distance entre le point non classifié et tous les autres points du dataset

LES K PLUS PROCHES VOISINS

Étape 1

- On choisit le nombre de K voisins les plus proches que l'on souhaite prendre en considération

Étape 2

- On calcule la distance entre le point non classifié et tous les autres points du dataset

Étape 3

- On sélectionne les K plus proches voisins. Parmi les K plus proches on compte combien de points sont attribués à chacune des classes

LES K PLUS PROCHES VOISINS

Étape 1

- On choisit le nombre de K voisins les plus proches que l'on souhaite prendre en considération

Étape 2

- On calcule la distance entre le point non classifié et tous les autres points du dataset

Étape 3

- On sélectionne les K plus proches voisins. Parmi les K plus proches on compte combien de points sont attribués à chacune des classes

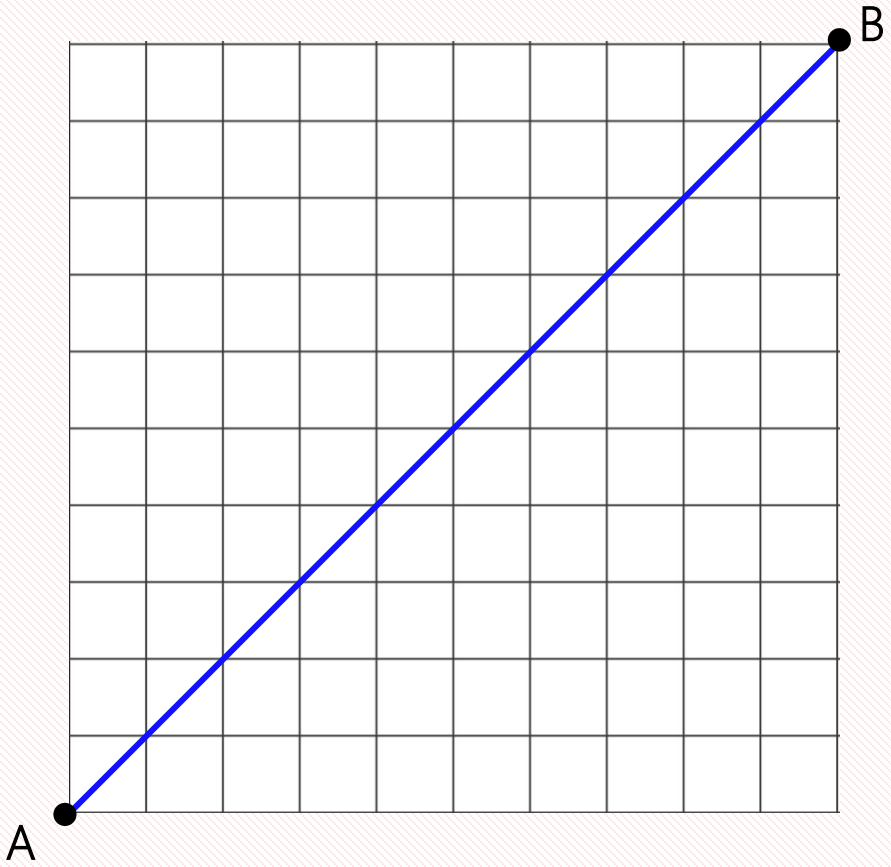
Étape 4

- On attribue la classe la plus fréquente à notre point

LES K PLUS PROCHES VOISINS

$$Distance\ euclidienne:$$

$$D_e(x,y) = \sqrt{\sum_{j=1}^{n}(x_j - y_j)^2}$$

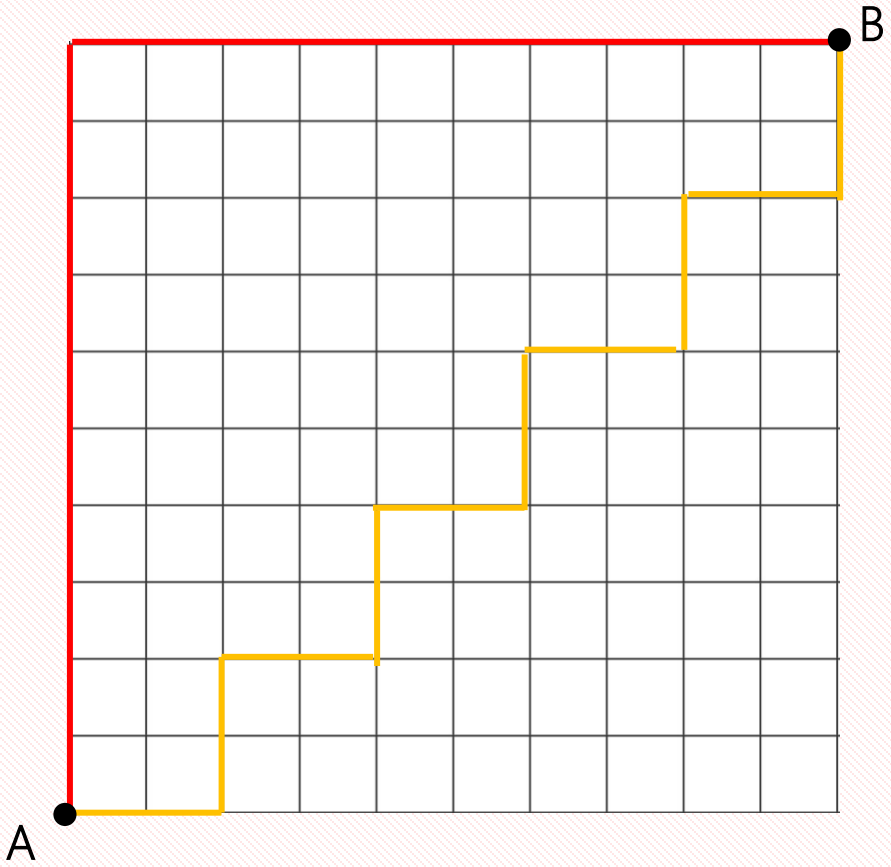


LES K PLUS PROCHES VOISINS

$$distance\ de\ Manhattan :$$

$$D_m(x,y) = \sum_{j=1}^{k} |x_i - y_i|$$

C'est la distance entre deux points lorsque l'on emprunte un chemin à travers un réseau ou quadrillage . Cette distance a été définie par Minkowski.



sklearn.metrics.DistanceMetric — scikit-learn 1.1.1 documentation

LES K PLUS PROCHES VOISINS

Metrics intended for real-valued vector spaces:

identifier	class name	args	distance function
“euclidean”	EuclideanDistance	•	`sqrt(sum((x - y)^2))`
“manhattan”	ManhattanDistance	•	`sum(|x - y|)`
“chebyshev”	ChebyshevDistance	•	`max(|x - y|)`
“minkowski”	MinkowskiDistance	p, w	`sum(w * |x - y|^p)^(1/p)`
“wminkowski”	WMinkowskiDistance	p, w	`sum(|w * (x - y)|^p)^(1/p)`
“seuclidean”	SEuclideanDistance	V	`sqrt(sum((x - y)^2 / V))`
“mahalanobis”	MahalanobisDistance	V or VI	`sqrt((x - y)' V^-1 (x - y))`

sklearn.metrics.DistanceMetric — scikit-learn 1.1.1 documentation

LES K PLUS PROCHES VOISINS

neighbors.BallTree(X[, leaf_size, metric])	BallTree for fast generalized N-point problems
neighbors.KDTree(X[, leaf_size, metric])	KDTree for fast generalized N-point problems
neighbors.KernelDensity(*[, bandwidth, ...])	Kernel Density Estimation.
neighbors.KNeighborsClassifier([...])	Classifier implementing the k-nearest neighbors vote.
neighbors.KNeighborsRegressor([n_neighbors, ...])	Regression based on k-nearest neighbors.
neighbors.KNeighborsTransformer(*[, mode, ...])	Transform X into a (weighted) graph of k nearest neighbors.
neighbors.LocalOutlierFactor([n_neighbors, ...])	Unsupervised Outlier Detection using the Local Outlier Factor (LOF).
neighbors.RadiusNeighborsClassifier([...])	Classifier implementing a vote among neighbors within a given radius.
neighbors.RadiusNeighborsRegressor([radius, ...])	Regression based on neighbors within a fixed radius.
neighbors.RadiusNeighborsTransformer(*[, ...])	Transform X into a (weighted) graph of neighbors nearer than a radius.
neighbors.NearestCentroid([metric, ...])	Nearest centroid classifier.
neighbors.NearestNeighbors(*[, n_neighbors, ...])	Unsupervised learner for implementing neighbor searches.
neighbors.NeighborhoodComponentsAnalysis([...])	Neighborhood Components Analysis.

neighbors.kneighbors_graph(X, n_neighbors, *)	Compute the (weighted) graph of k-Neighbors for points in X.
neighbors.radius_neighbors_graph(X, radius, *)	Compute the (weighted) graph of Neighbors for points in X.

Pour l'apprentissage supervisé

LES K PLUS PROCHES VOISINS

#les hyperparamètres pour l'algorithme KNeighborsClassifier ou Regressor :

- n_neighbors : nombre de points que l'on souhaite prendre en considération
- weights : poids accordé à chaque points
  - 'uniform' : tous les points ont le même poids peu importe leur distance
  - 'distance' : le poids attribué dépend de leur distance plus les points sont éloignés plus leur poids est faible.
- P : paramètre utilisé dans la métrique de Minkowsky . Si p= 2 correspond à la distance euclidienne et p=1 on utilise la distance de Manhattan
- metric: par défaut c'est celle de Minkowski

#les hyperparamètres pour l'algorithme RadiusNeighborsClassifier:

Ce sont les même que précédemment mais à la place du nombre de voisins on peut définir

- radius: le rayon , par défaut il est à à 1

LES K PLUS PROCHES VOISINS

Avantages

- L'algorithme est **simple et facile à mettre en œuvre**.
- Il n'est pas nécessaire de créer un modèle, de régler plusieurs paramètres ou de formuler des hypothèses supplémentaires.
- L'algorithme est polyvalent. Il peut être utilisé pour la **classification** ou la **régression**.

Inconvénients

- L'algorithme devient beaucoup plus lent à mesure que le nombre d'observations et de variables indépendantes augmente.