INU Champollion

Licence Info 3ème Année

Maths pour l'I.A.

Thierry Montaut

2021

Rappel : Apprentissage supervisé

- On parle d'apprentissage supervisé lorsque la méthode d'apprentissage de la fonction f nécessite l'utilisation d'une base de données pour lesquelles ont connaît exactement le résultats. Dit autrement, d'un ensemble de couples $\{(x_1, y_1), ...(x_n, y_n)\}$ pour lesquels $y_i = f(x_i)$.
- L'apprentissage supervisé est l'un des types les plus courants et les plus efficaces d'apprentissage automatique.

Rappel : Différents problèmes d'apprentissage

On peut toujours considérer qu'à partir d'un ensemble X, on cherche à apprendre pour tout x dans X la valeur $y = f(x)$ que prend une fonction f en x.
Quand on cherche à apprendre la fonction

$$f : \begin{pmatrix} X & \to & Y \\ x & \mapsto & y = f(x) \end{pmatrix}$$

On distingue les problèmes d'apprentissage selon la nature de l'ensemble Y :

- Si Y est un ensemble fini, on parle de problème de **classification**. La fonction à prédire est un classificateur.
- Si $Y \subseteq \mathbb{R}$, on parle de problème de **régression**.

Quelques classificateurs et une régression

Nous allons aborder dans ce chapitre quelques méthodes de classification supervisée et une régression linéaire

1. l'Algorithme k-NN des k plus proches voisins
2. La régression linéaire par la méthode des moindres carrés.
3. Un classificateur linéaire élémentaire par correction d'erreur
4. Un classificateur linéaire à descente de gradient
5. Un Séparateur à Vaste Marge (SVM)

Ce qui va commencer à nous familiariser avec certains concepts et termes clés de l'apprentissage automatique.
Leur étude se poursuivra au S5.

Généralisation et surapprentissage

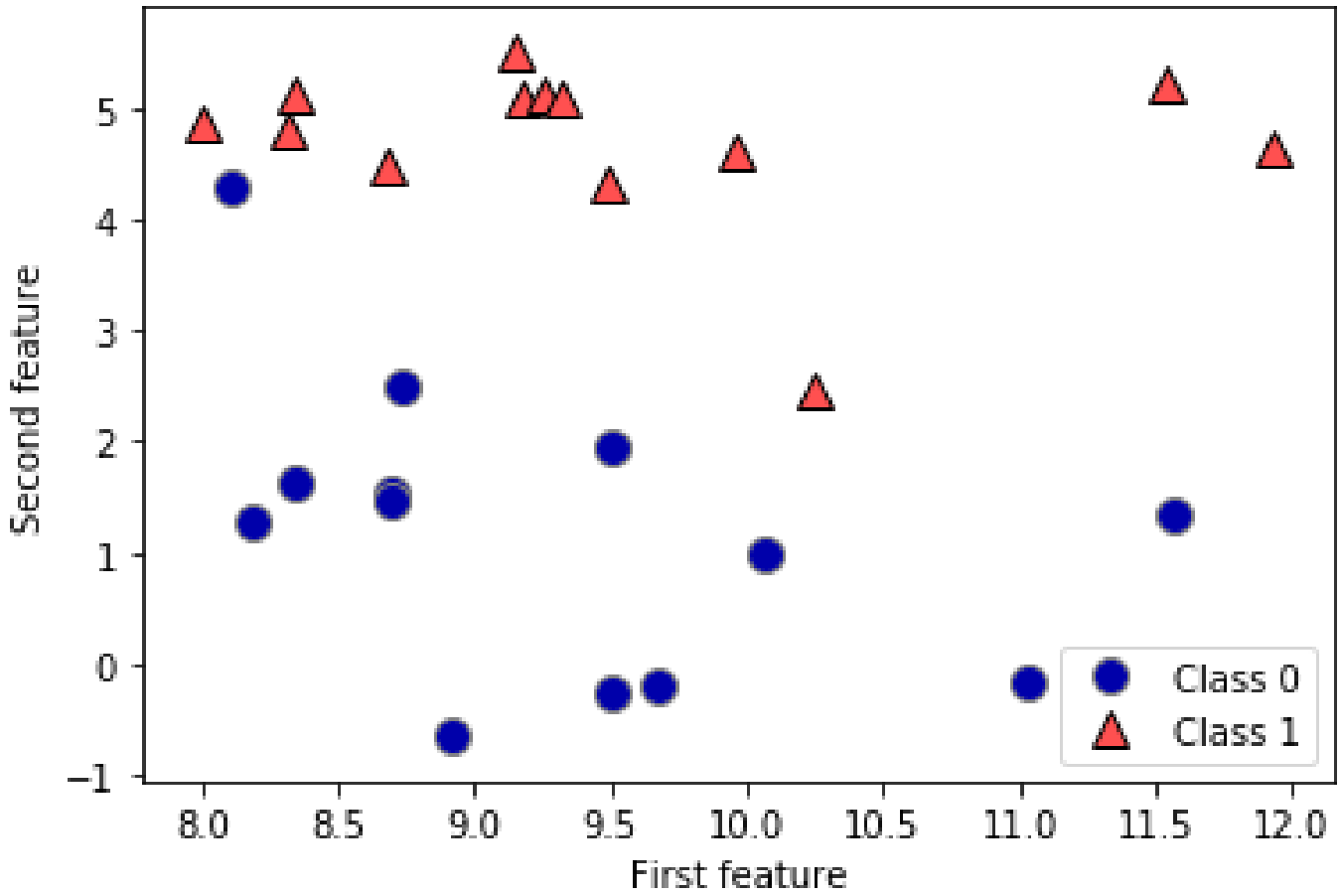
1. Dans l'apprentissage supervisé, nous souhaitons construire un modèle à partir d'un jeu de **données d'apprentissage** pour lesquelles on connaît la valeur de f, puis être capable d'en déduire la valeur de f sur des **données de test** pour lesquelles on ne connaît pas le résultat.
2. La première phase est **l'apprentissage du modèle**, la seconde est la phase de **généralisation ou de prédiction**.
3. Un bon modèle n'est pas un modèle qui fonctionne bien sur les données d'apprentissage mais qui se généralise bien à de nouvelles données de test.

Généralisation et surapprentissage

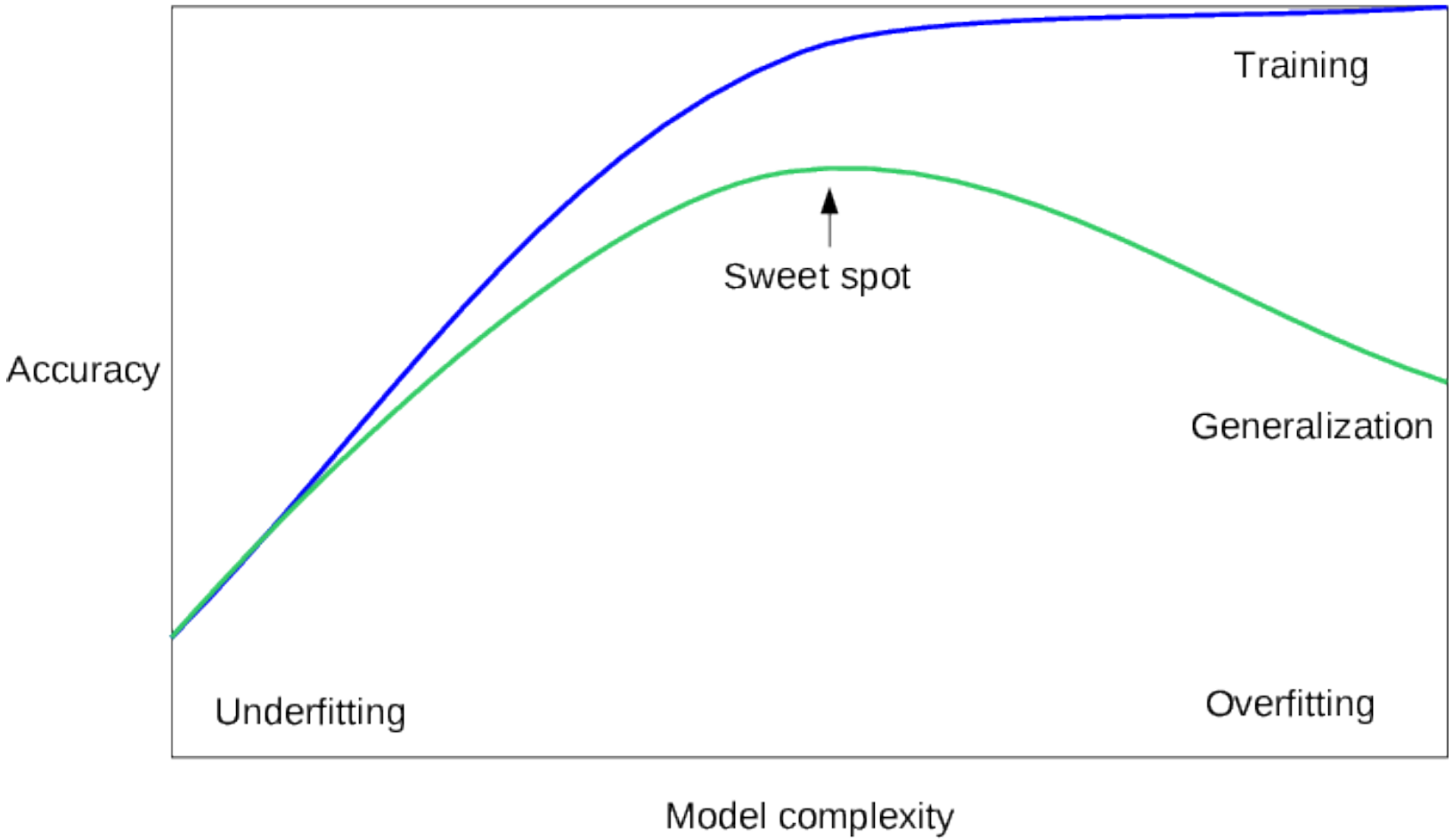
1. On commencera donc toujours pas scinder aléatoirement les données dont on dispose en deux (souvent 75% pour l'apprentissage et 25% pour le test). Les performances seront calculées sur **le taux de bonnes prédiction sur les données de test.**
2. On peut être tenté de construire des modèles complexes collant parfaitement aux données d'apprentissage (même si elle contiennent des données exubérantes et rares (il est toujours possible d'être aussi exacte que l'on souhaite sur les données d'apprentissage). Mais cela se fait souvent au détriment de la capacité de généralisation à de nouvelles données. C'est le phénomène de **surapprentissage.**

Surapprentissage

Comment classifier au mieux le jeu de donné *forge* ?



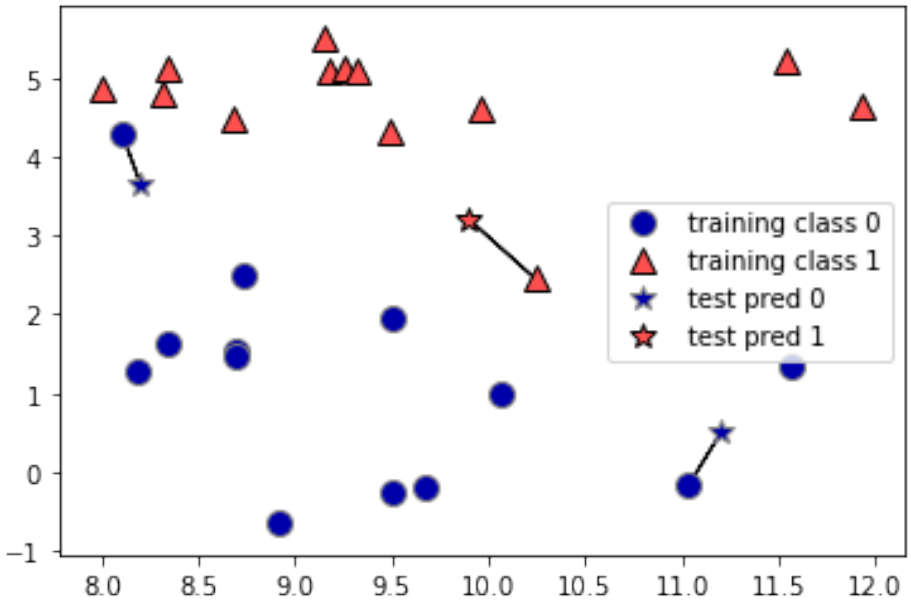
On cherche donc un compromis entre la complexité du modèle lors de l'apprentissage et sa capacité de généralisation, ce qui nécessite une phase d'optimisation.

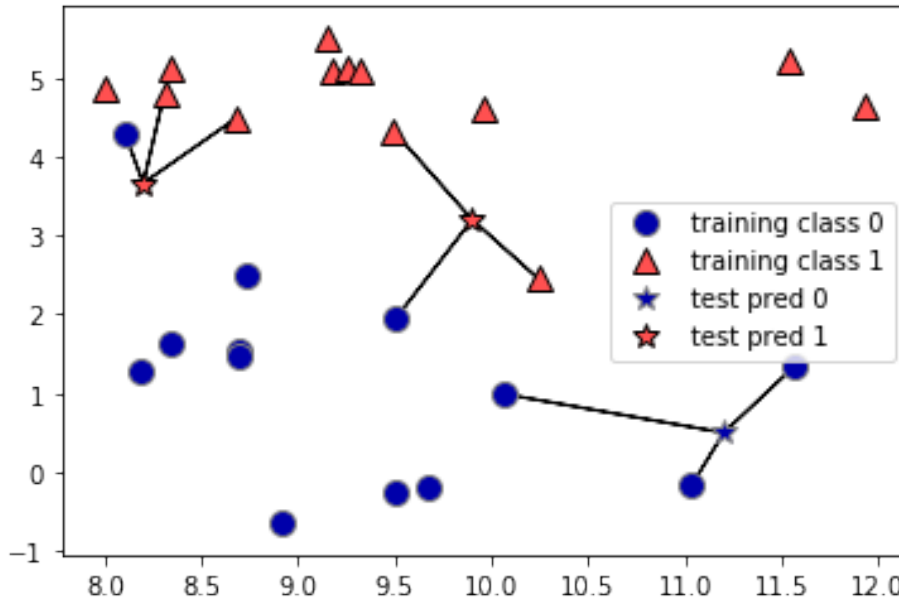


Les k plus proches voisins

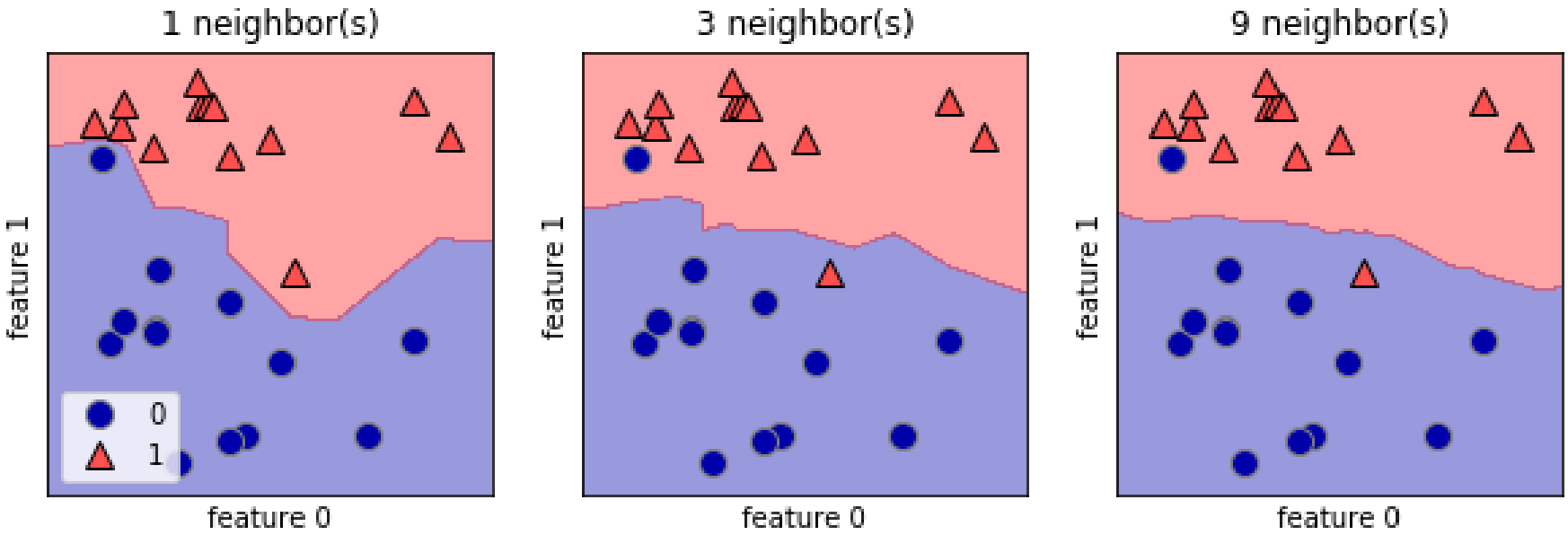
- L'algorithme k-NN des k plus proches voisins est le plus simple des apprentissages automatiques. Il n'y a aucun paramètre à apprendre. L'apprentissage se résume à choisir le jeu de données d'apprentissage A et à choisir une valeur de $k \geq 1$.
- Prédiction : étant donné une nouvelle donnée $x \in X$, on calcule les distances (euclidiennes) de x à tous les éléments de A et on sélectionne les k plus proches. On attribue à x la valeur majoritaire parmi ses k plus proches voisins.
- Nous mettrons en oeuvre ce modèle en TP à l'aide du jeu de données *forge* de la librairie scikit-learn de Python.

k-NN pour $k = 1$ et $k = 3$.

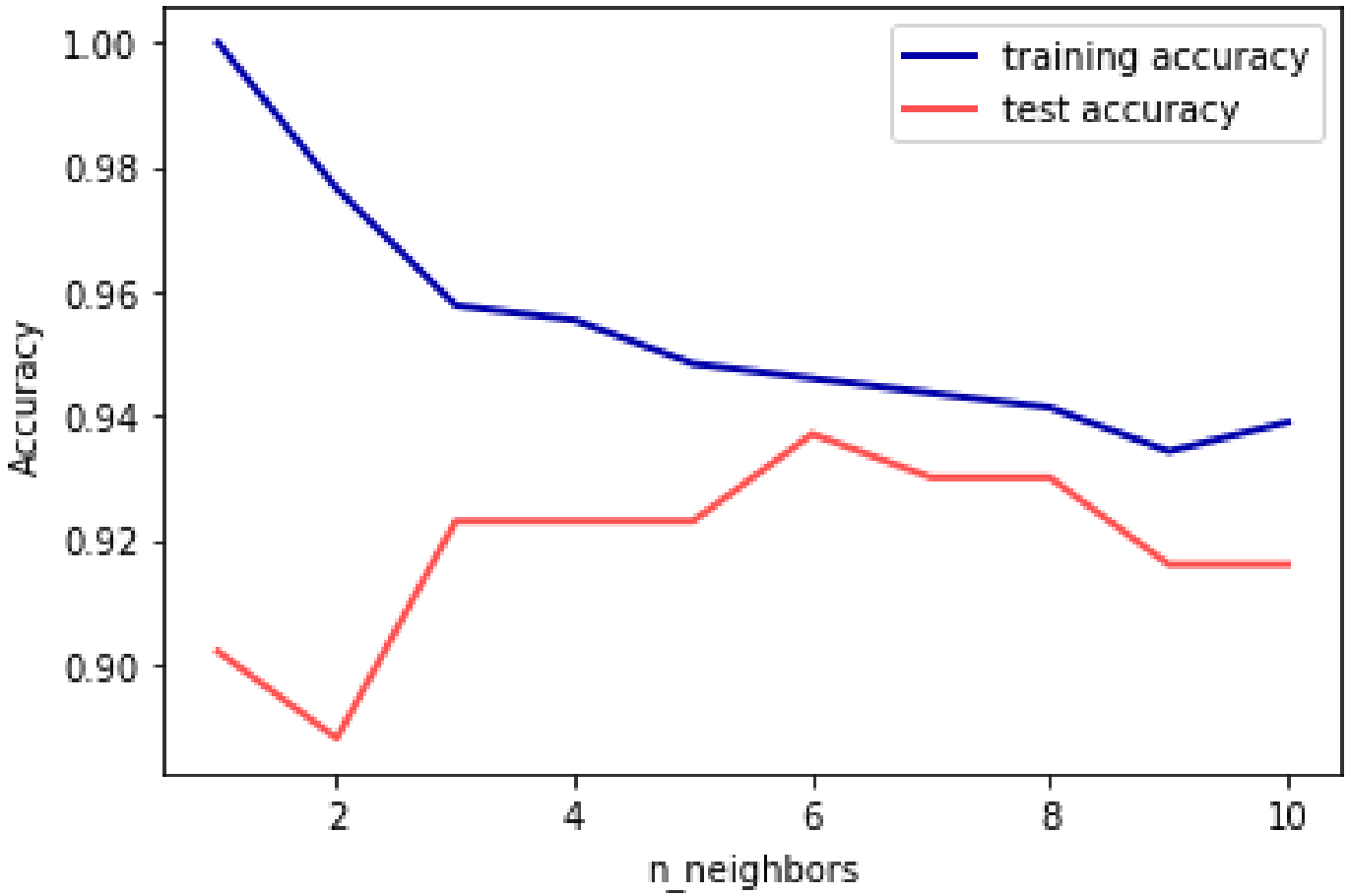




Pour différentes valeurs de k



surapprentissage et apprentissage optimal

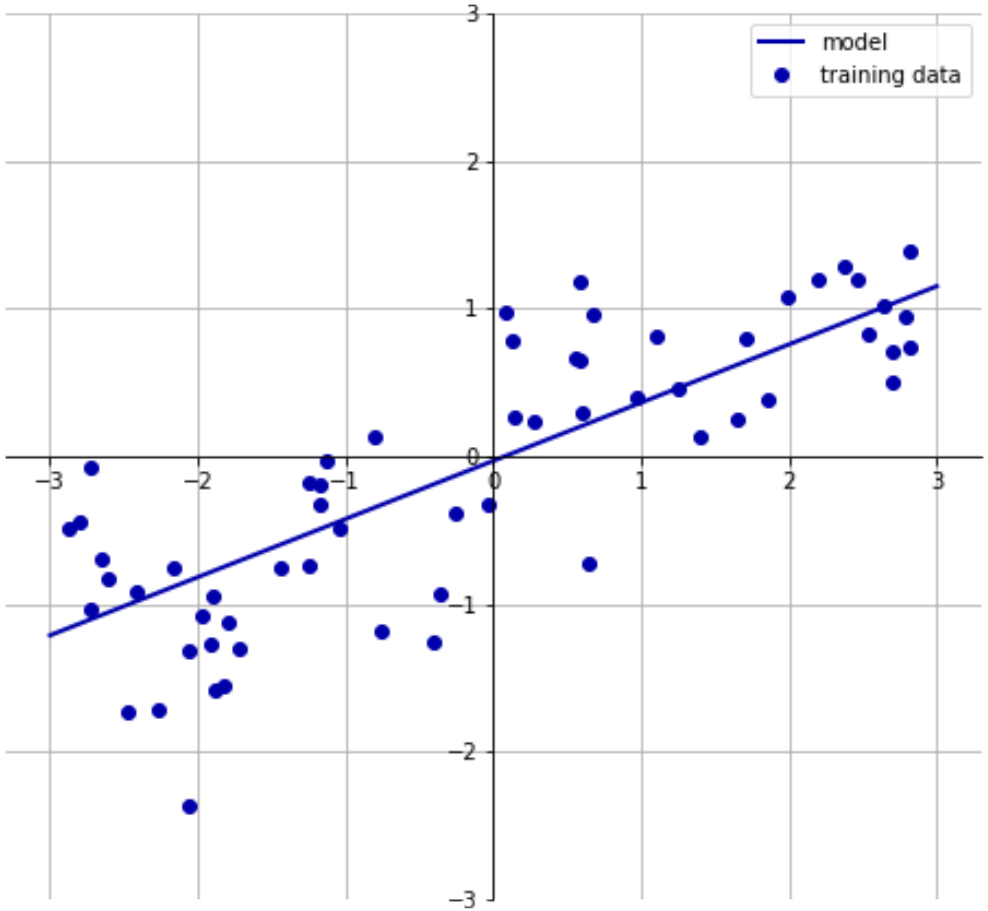


Paramètres

1. Le modèle k-NN possède deux paramètres importants : le nombre k de voisins et la distance choisie.
2. En pratique se limiter à un petit nombre de voisins proche de 5 donne souvent de bons résultats mais il est bon d'ajuster ce paramètre comme précédemment à l'aide du jeu de données fourni.
3. Nous n'utiliserons cette année que la distance euclidienne, mais vous découvrirez en master des distances dans $\mathbb{R}^n$ pouvant donner de meilleures performances.

Exemple de régression linéaire

- Ce sont les plus anciennes des méthodes de prédiction.
- Cette fois-ci, toutes les valeurs réelles sont possibles pour $f(x)$.
- L'idée générale est de chercher à déterminer, dans le cas de $\mathbb{R}^2$, une droite d'équation $y = ax + b$ la plus proche possible des données (x_a, y_a) d'apprentissage, et qui permettra la prédiction $f(x) = ax + b$.
- Plus généralement dans $\mathbb{R}^n$, on cherche l'équation d'un hyperplan H d'équation $a_1 x_1 + a_2 x_2 + \ldots + a_n x_n + b = 0$, le plus proche possible des données d'apprentissage $(x_1, \ldots, x_n, y)$ et on effectuera la prédiction $f(x) = a_1 x_1 + a_2 x_2 + \ldots + a_n x_n + b$.


model
training data
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

Apprentissage du modèle

1. Dans le cas de $\mathbb{R}^2$, on calcule la somme des distances au carré des points d'apprentissage à la droite d'équation $y = ax + b$. Cela donne une fonction de deux variables $d(a, b)$. Il s'agit de déterminer le couple (a, b) minimisant cette fonction.
2. Dans le cas de $\mathbb{R}^n$, il s'agit de déterminer $(a_1, \ldots, a_n, b)$ minimisant la distance des points à l'hyperplan H.
3. On est donc ramené à la minimisation d'une fonction de plusieurs variables, ce qu'on ne sait pas encore faire...
4. Nous attendrons donc le S5 pour résoudre théoriquement ce problème, mais nous pourrons l'illustrer en TP à l'aide de la fonction LinearRegression() de la bibliothèque scikit-learn.

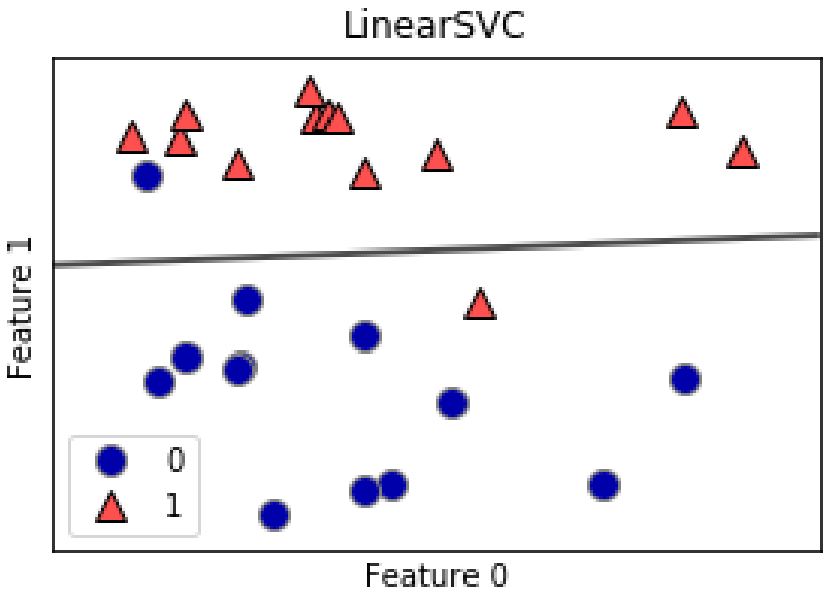
Classificateurs linéaire

- Dans $\mathbb{R}^2$, un classificateur linéaire cherche à déterminer la droite qui sépare au mieux notre jeu de données.
- Plus généralement si nos données possèdent n caractéristiques et non seulement deux, nous cherchons dans $\mathbb{R}^n$ l'équation d'un hyperplan (un sous-espace vectoriel de dimension $n-1$) séparant au mieux nos données.
- un classificateur linéaire (binaire) est un classificateur qui sépare deux classes à l'aide d'un hyperplan d'équation

$$a_1 x_1 + \ldots + a_n x_n + b = 0.$$

- Il s'agit donc de déterminer les $n+1$ coefficients $(a_1, \ldots, a_n, b)$ de manière à classer au mieux les données d'apprentissage. On est dans la classe 1 si $a_1 x_1 + \ldots + a_n x_n + b \geq 0$ et dans la classe 2 sinon. Donc si $w = (a_1, \ldots, a_n)$,

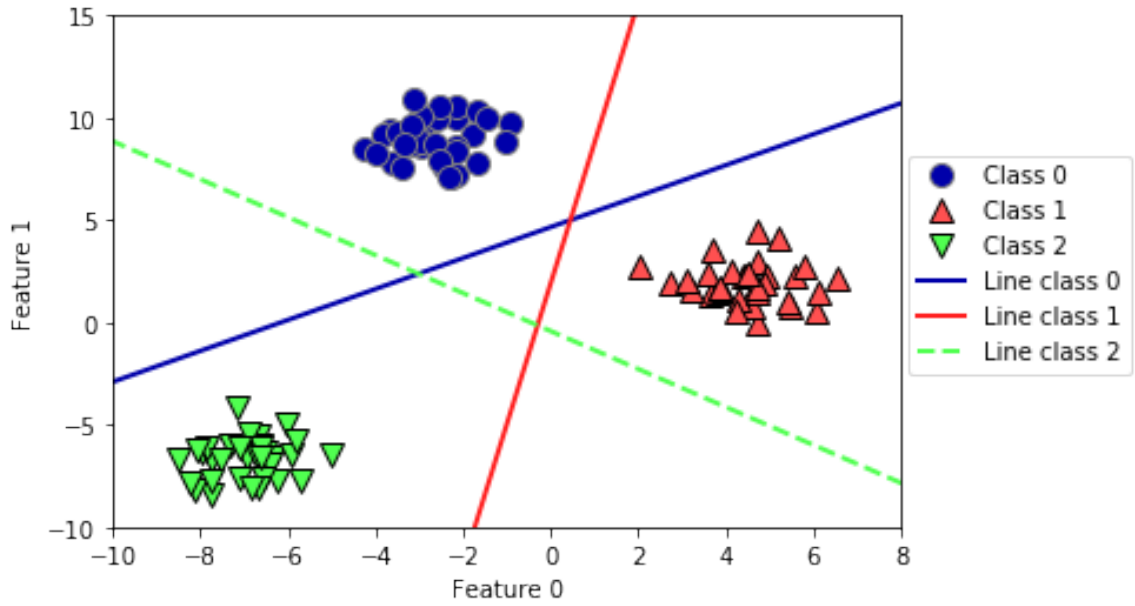
$$f(x) = signe((w|x) + b).$$


LinearSVC
Feature 1
Feature 0
0
1


LogisticRegression
Feature 1
Feature 0

Cas de *n* classes

On détermine alors *n* hyperplan suivant la stratégie "seul contre tous" en isolant une classe de toutes les autres par un hyperplan :

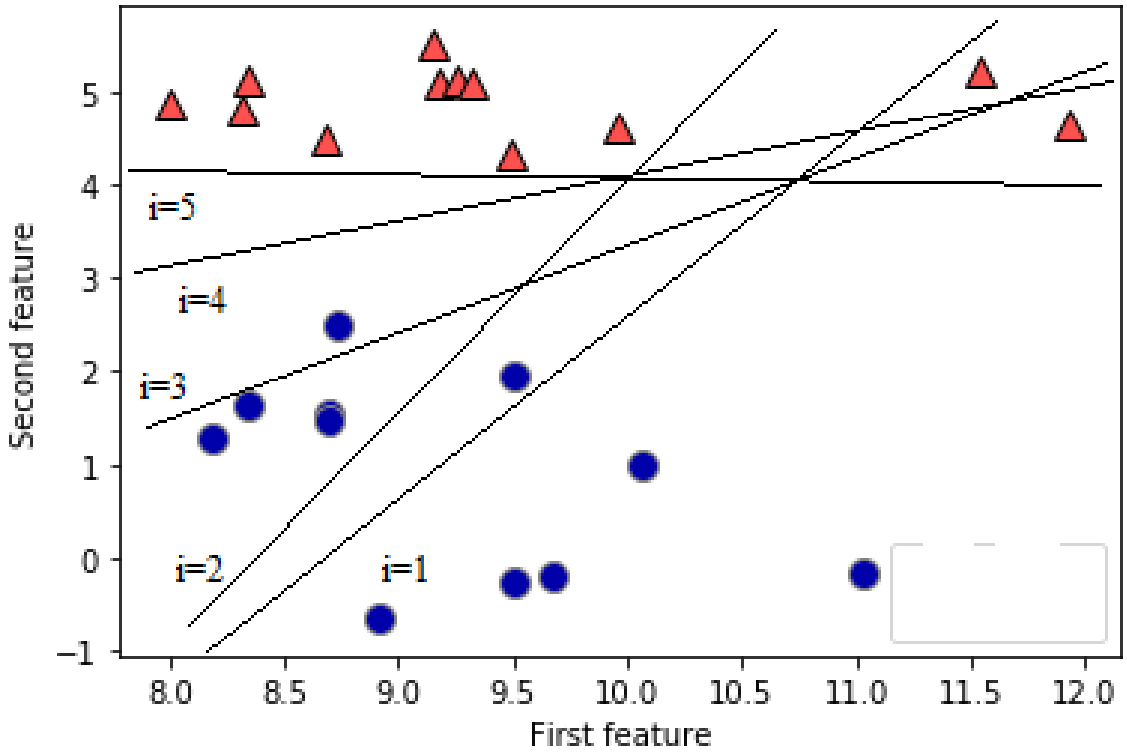


Par correction itérative d'erreur

L'idée de ce premier algorithme (glouton) est simple :

```
On choisit arbitrairement ou aléatoirement
  un premier hyperplan.
Pour toute donnée d'apprentissage (x,y) dans A :
    calculer d=y-signe((w|x)+b)
    # d est nul si l'hyperplan actuel prédit bien x
    # et vaut 2 ou -2 sinon
    si d!=0:
        # on corrige w et b pour que x soit bien prédit
      ... à réfléchir!
On itère jusqu'à ce que tous les points
  soient bien prédits.
```

Correction itérative d'erreurs



Exercice

- Essayer d'apprendre de cette manière la fonction booléenne "or" à partir des données :

$$[[(0,0),-1],[(0,1),1],[(1,0),1],[(1,1),1]]$$

- Apprendre à classifier dans $\mathbb{R}^2$, $A_1 = \{(x,y), x < 2y\}$ et $A_{-1} = \{(x,y), x \geq 2y\}$ à l'aide des données

$$[[(0,2),1],[(1,1),1],[(1,2.5),1],[(2,0),-1],[(3,0.5),-1]].$$

Remarques sur cette méthode

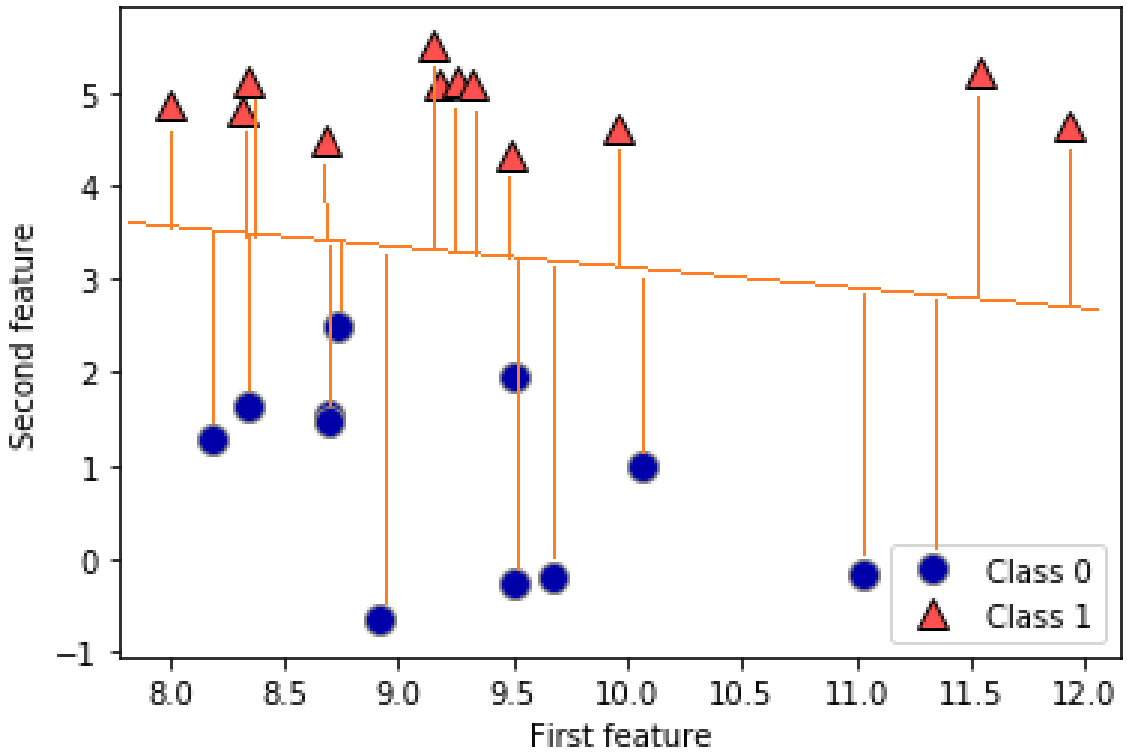
1. La question de la convergence de la méthode est ardue. En particulier, elle ne converge pas si les données d'apprentissage ne sont pas linéairement séparables. On peut démontrer qu'elle converge dans le cas contraire.
2. Même lorsqu'elle converge, rien n'assure qu'elle permettra de bonnes prédictions sur de nouvelles données. Comme on le voit sur l'illustration le séparateur obtenu n'est pas optimal et ne résisterait pas à de nouvelles données de test.

Droite de meilleur approximation

- Plutôt que de minimiser le nombre d'erreurs d'apprentissage, donc une valeur entière, on va introduire une erreur continue pour bénéficier des méthodes d'optimisation de l'analyse réelle.
- On ne regarde donc plus seulement si la prédiction est bonne mais la distance à la prédiction et plus généralement la somme des carrés des distances à la prédiction.
- On définit donc l'erreur :

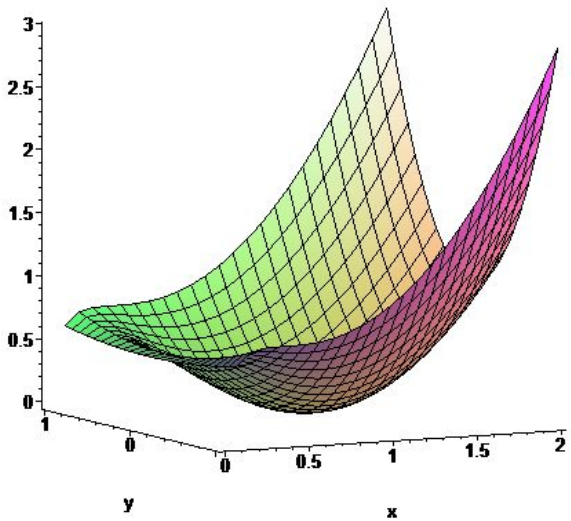
$$E(w,b) = \sum_{(x,y)\in A} ((w|x) + b - y)^2.$$

- On cherche la droite qui minimise cette erreur.


Second feature
First feature
Class 0
Class 1
5
4
3
2
1
0
−1
8.0
8.5
9.0
9.5
10.0
10.5
11.0
11.5
12.0

Par descente de gradient

- $E(w, b)$ est une fonction de plusieurs variables que l'on cherche à minimiser ce que nous ne savons pas encore faire...
- Nous pourrons utiliser une méthode numérique de "descente de gradient" pour obtenir ce minimum

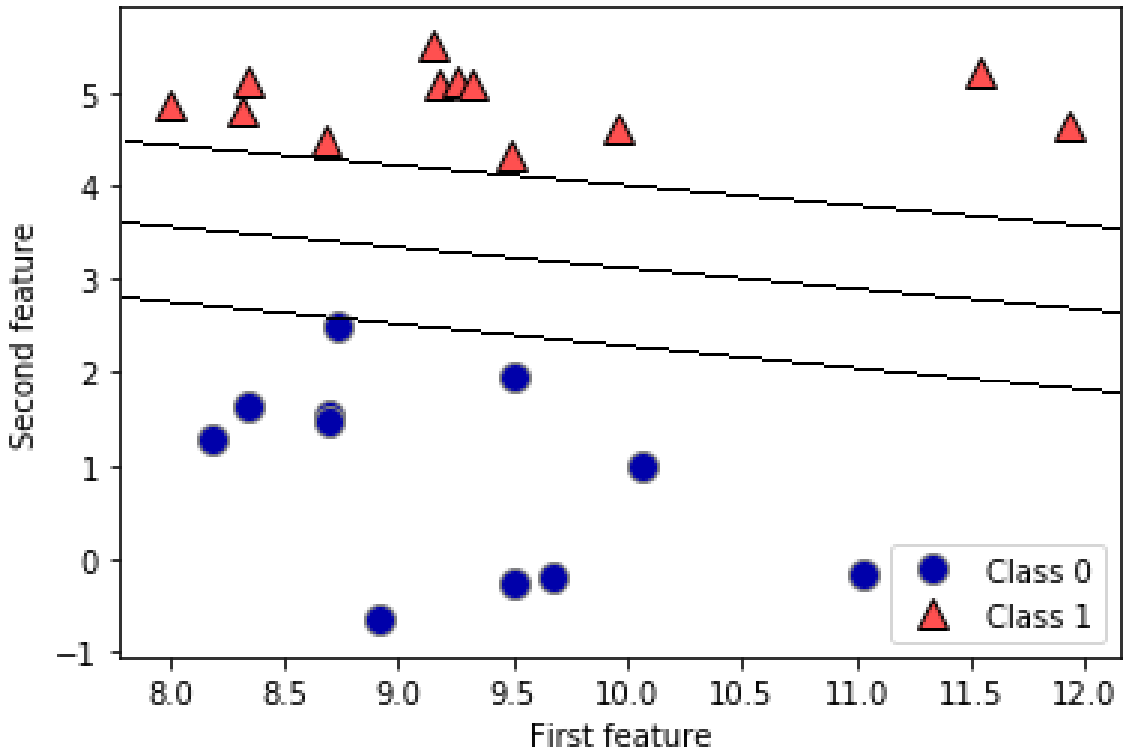


Séparateurs à Vaste Marge

- Les séparateurs à vaste marge (SVM, Support Vector Machine en anglais) ont été mis au point à partir des années 2000 par des études théoriques en statistique inférentielle. Ce sont les algorithmes actuels de séparation linéaire les plus élaborés mathématiquement mais surtout les plus efficaces.
- Leur succès tient également au fait qu'on sait les adapter à des séparations non linéaires.
- On a vu que l'algorithme de minimisation de l'erreur quadratique est plus efficace que la correction itérative d'erreurs mais il est encore sensible aux valeurs extrêmes (les points éloignés de la zone de séparation).
- Au contraire les SVM vont essayer d'optimiser cette zone de séparation sans se soucier des points qui en sont éloignés.

Séparateurs à Vaste Marge

- L'idée générale est de déterminer un parallélogramme de surface maximale séparant les points d'apprentissage et de considérer comme droite de séparation, le milieu de ce parallélogramme.
- Le parallélogramme est déterminé par 2 vecteurs dans $\mathbb{R}^2$ (et c'est un hyper-parallélépipède déterminé par n vecteurs en dimension n.). On les appelle les vecteurs de support, ce qui justifie le nom anglais de la méthode)
- On cherche donc maintenant n vecteurs tels que le parallélogramme défini par ces vecteurs soit séparateur des points d'apprentissage et de surface maximal.
- On est donc encore ramené à un problème de calcul de maximum d'une fonction de plusieurs variables.


Second feature
First feature
Class 0
Class 1
5
4
3
2
1
0
−1
8.0
8.5
9.0
9.5
10.0
10.5
11.0
11.5
12.0

- Il est donc temps d'en apprendre davantage sur les fonctions de plusieurs variables, leurs dérivées, leurs variations et leurs extrema.
- En attendant nous pourrons utiliser et illustrer ces méthodes en TP à l'aide des fonctions prédéfinies des bibliothèques python scikit-learn ou tensorFlow...
- ... et vous pouvez déjà commencer à vérifier que vous savez dériver et étudier les variations des bonnes vieilles fonctions réelles.