Classification Linéaire

Cette vidéo résume super bien les SVM en 1h : https://www.youtube.com/watch?v=_PwhiWxHK8o

Données linéairement séparables, Perceptron

Essaye de créer un hyperplane qui sépare les données.

On minimise :

$$E(w) = \sum_{n=1}^{N} L\left(y\left(x_n; w\right), t_n\right)$$

Algorithme :

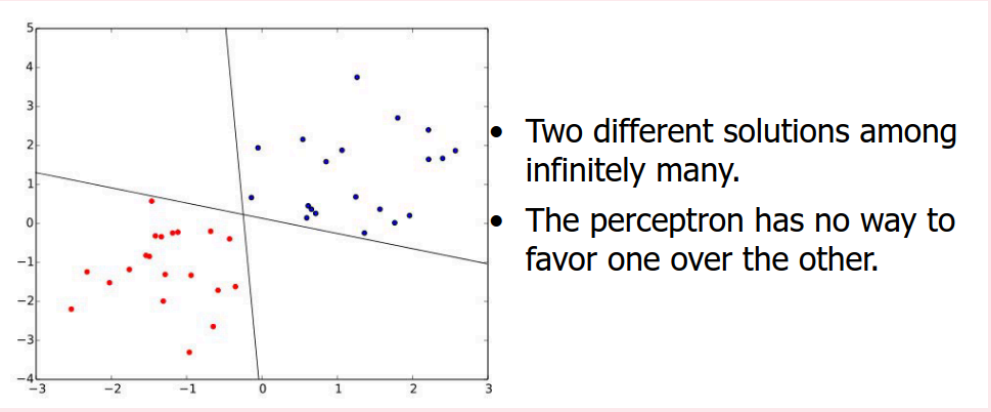
- On définit $w_{t=1}$ à un vecteur de zéro.
- Itérativement :
  - si x_n est correctement classifié, on ne fait rien.
  - sinon, $w_{t=t+1} = w_t + t_n x_n$, $b_{t+1} = b_t + t_n$

Une fois qu'on a w, comment décider ?

Une fois qu'on a l'équation du plan $\vec{w} \cdot \vec{x} + b = 0$, avec $\vec{w}$ le vecteur perpendiculaire au plan, $\vec{x}$ notre point. Ensuite $f(x) = \text{sign}(\vec{w} \cdot \vec{x} + b)$ soit $\hat{y} = +1$ si $\vec{w} \cdot \vec{x} + b > 0$ sinon $\hat{y} = -1$.

On obtient donc une fonction de décision $f(x) = \vec{w} \cdot \vec{x} + b$ donc on regarde le signe pour classifier. C'est une **step function** (elle passe de 0 à 1 d'un coup).

1er problème : pas d'optimisation de la marge



Le perceptron n'a aucune idée de qui est le meilleur.

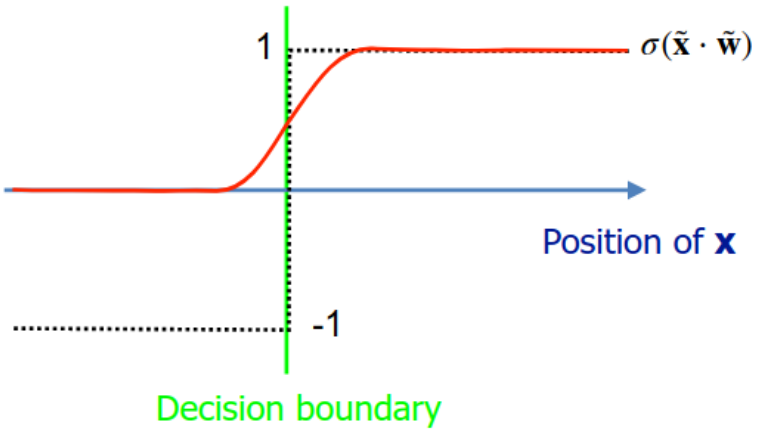
Régression Logistique

Un autre problème avec le perceptron est qu'on a aucune façon de dire si on est proche ou pas de la frontière de décision.

- on remplace la step function par une sigmoïde, le modèle produit une probabilité $\hat{p} \in (0, 1)$.
- maintenant, on minimise la **cross-entropy** (log-loss) sur le jeu de données $\{(x_n, t_n)\}$, où $t_n \in \{0, 1\}$:

$$L_{\mathrm{CE}}(w) = -\sum_{n=1}^{N} \Big[t_n \, \log\big(\hat{p}_n\big) \; + \; (1 - t_n) \, \log\big(1 - \hat{p}_n\big) \Big].$$

(plus stable, deux fois différentiable, etc.)



- comme on utilise cette cross-entropy loss, si le plan passe très près de certains points, leur probabilité d'appartenir à une classe ou l'autre va être p. exemple 60/40 du coup le modèle va être loin de la **vraie** prédiction (100/0), ce qui va quand même pénaliser le modèle, alors que le perceptron aurait été content. cela pousse le modèle à augmenter la taille de la marge.

Un autre problème : les outliers

On n'a pas introduit de slack variables, les outliers pénalisent beaucoup la logistic regression sur le test set.

Vers le multi-class

On passe de la sigmoid vers la softmax, la même chose mais généralise l'idée quand on veut que K classes, sommées, donnent 1.

Max Margin Classifier

On cherche à maximiser la marge, calcul de la distance d'un point à un hyperplan

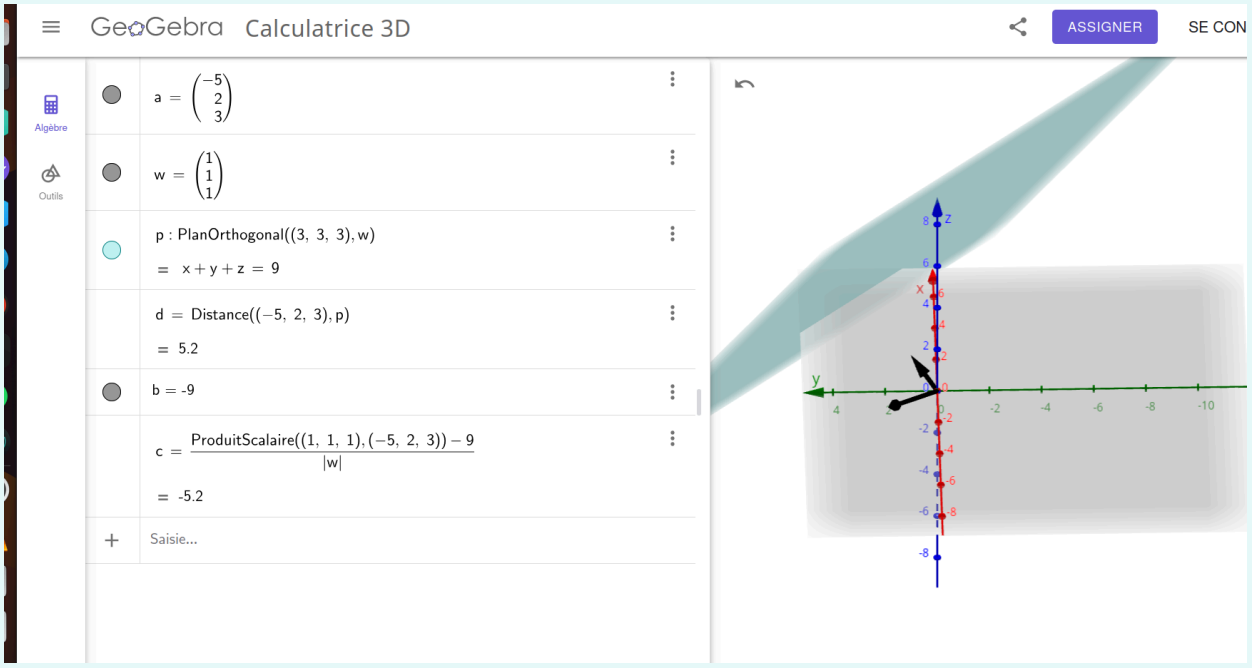
La marge est définie comme

$$\text{marge } = \min_i \left(\text{distance de } x_i \text{ à l'hyperplan}\right) \cdot I\left(x_i \text{ bien classé}\right)$$

Comment calculer la distance d'un point $\vec{x}$ à un hyperplan de vecteur normal $\vec{w}$ et de biais b ?

On projette donc $\vec{x}$ sur $\frac{\vec{w}}{||\vec{w}||}$ pour obtenir sa composante selon $\vec{w}$ unitaire : $\vec{x} \cdot \frac{\vec{w}}{||\vec{w}||}$. On doit aussi ajouter le bias :

$$d\left(x, \text{ plan}\right) = \left| \left(\frac{w \cdot x + b}{\|w\|} \right) \right|$$



Pour trouver b, si on sait que notre plan passe par le point $(3, 3, 3)$, on fait $w^T(3, 3, 3) + b = 0$, comme le point vérifie l'équation, et ainsi on trouve b.

On veut donc :

$$\max(\text{"marge"}) = \max_{w,b} \left(\min_i \frac{y_i(w \cdot x_i + b)}{\|w\|} \right)$$

Pour simplifier ce problème d'optimisation avec min-max :
On introduit la marge **fonctionnelle** :

$$\gamma_i^f = y_i \left(w \cdot x_i + b\right)$$

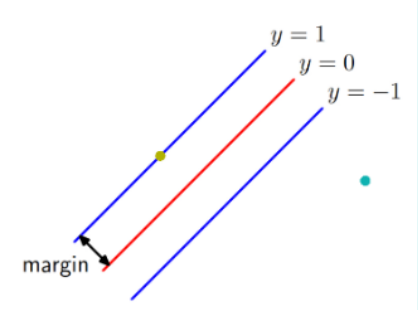
si $\gamma_i^f > 0$ le point est bien classé (y_i étant le vrai label $+1$ ou -1), et plus γ_i^f est grand, plus la certitude est grande.

Et la marge **géométrique** réelle :

$$\gamma_i^g = \frac{\gamma_i^f}{\|w\|}$$

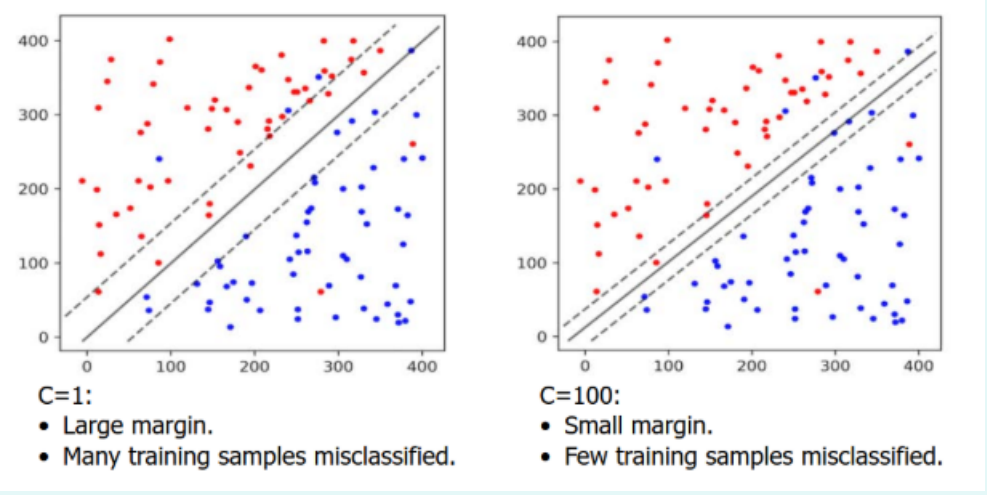
C'est la distance signée au plan (positive si point bien classé), et la marge du classifieur est typiquement $\min_i \gamma_i^g$.

Lorsque l'on fixe la marge fonctionnelle a 1 par la contrainte, alors la marge géométrique minimale vaut $\frac{1}{\|w\|}$ (et la totale, des deux côtés $\frac{2}{\|w\|}$. D'ou l'idée de maximiser $\frac{1}{\|w\|}$, ou de minimiser $\frac{1}{2}\|w\|^2$ (plus simple pour les calculs).

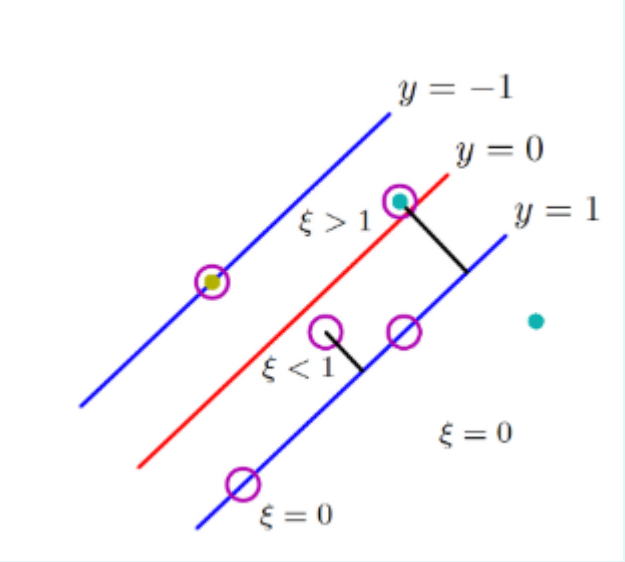


Le problème, c'est qu'on veut toujours que certains points puissent violer la marge (le problème ne doit pas être parfait). Pour ça, on ne dit plus que $y_i \left(w^\top x_i + b\right) \geq 1$ mais que $y_i \left(w^\top x_i + b\right) \geq 1 - \xi_i$ et veut minimiser :

$\text{Loss } = \frac{1}{2}\|w\|^2 + C \sum_i \xi_i$



Pour que le modèle prenne à la fois en compte le fait qu'on ne doit pas missclassifier des points, mais qu'on doit aussi que ceux bien classifiés doivent être le plus possibles.



- Si $\xi_i = 0$, le point est bien classifié et en dehors de la marge.
- Si $0 \leq \xi_i \leq 1$, le sample est classifié et dans la marge
- Si $\xi_i \geq 1$, alors le sample est mal classifié.

Avec un grand C, on empêche donc les mauvaises classifications.

⑦ Et comment faire l'entraînement ?

On obtient donc cette fonction à minimiser $\text{Loss}\,(w) = \frac{1}{2}\|w\|^2 + C\sum_i \xi_i$ sous la contrainte $y_i\left(w^\top x_i + b\right) \geq 1 - \xi_i$.

Simplifions-là en enlevant les slack variables et calculons le Lagrangien ($\mathcal{L}(x, y, \lambda) = f(x, y) - \lambda g(x, y)$, puis $\mathcal{L}_{x,y}(x, y, \lambda) = 0$). On obtient donc :

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{N} \alpha_i \left[y_i\left(w^T x_i + b\right) - 1\right]$$

On dérive par rapport à w, b:

$$\frac{\delta \mathcal{L}}{\delta w} = 0 \Rightarrow w = \sum_i \alpha_i y_i x_i \text{ et } \frac{\delta \mathcal{L}}{\delta b} = 0 \Rightarrow \sum_i \alpha_i y_i = 0$$

On a trouvé la forme de notre w !

$$f(x) = w^T x + b = \sum_{i \in \text{ support vectors}} \alpha_i y_i x_i x + b$$

(voir la kernel matrix pour savoir comment trouver les alpha)

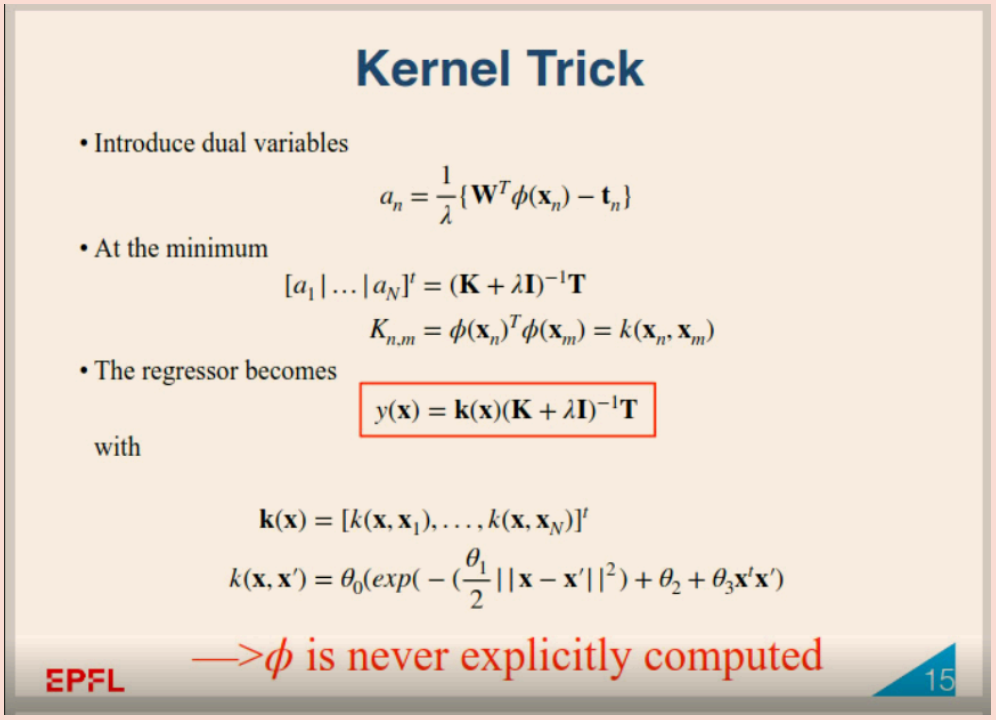
a l'inférence avec les variables slack :

$$f(x) = \sum_{i \in \text{ support vectors}} \alpha_i y_i k\left(x, x_i\right) + b \text{ avec } 0 \leq \alpha_i \leq C, \forall i$$

En fait les support vectors c'est les vecteurs qui déterminent la marge. Pour déterminer le label du nouveau point, on va calculer $k\left(\text{nouveau point}, \text{support vector}_i\right)$, sa similarité au support vector. S'il est très près, alors il a probablement le même label.

Kernel Matrix

Pour trouver les α, on utilise la Kernel Matrix.



On a donc les $\alpha = (K + \lambda I)^{-1}T$. λ est un hyperparamètre.

Classification non linéaire, comment on fait ? parfois les données ne sont pas séparables

Données non linéairement séparables

L'idée : augmenter les dimensions

Ce qu'on fait donc, c'est de choisir une fonction non linéaire, de l'appliquer à nos données (en fait au lieu d'avoir juste une série de points (x, y) on va avoir $(x, y, f(x, y))$, et d'espérer qu'elles soient séparables.

Polynomial expansion

On ne donne plus uniquement x mais $x_1, x_1^2, ..,$ etc. pour permettre une approximation de notre fonction avec un polynôme.

mais du coup on doit faire le produit scalaire entre $\Phi(x)$ et $\Phi(y)$, c'est très long. (rien que de calculer $\Phi(x)$ c'est long).

Régularisation : On ajoute souvent un terme $\frac{\lambda}{2}\|w\|^2$ pour éviter le sur-apprentissage :

$$L_{\text{CE+reg}}(w) = L_{\text{CE}}(w) \; + \; \frac{\lambda}{2}\|w\|^2.$$

On peut trouver le λ idéal avec la cross-validation. En fait avoir un grand w c'est un problème, parce que ça veut dire que certaines features ont une importance p. exemple énorme, et si on donne un nouveau point de test x qui a cette coordonnée un tout petit peu différente alors le résultat va être aussi différent.

SVM / Kernel trick

Le problème sans le Kernel Trick

Ça marche bien, mais c'est très long, d'autant plus que dans certains cas on a besoin d'énormément de dimensions pour séparer les données. Et c'est aussi difficile de trouver cette fonction Φ qui fait un bon mapping (il faut en tester plusieurs -- pas de recette magique).

Le kernel trick

Soit Φ notre fonction de mapping.
En fait, comme on le voit plus haut, notre fonction ressemble à ça :

$$f(x) = w^T\Phi(x) + b = \sum_{i=1}^{N} \alpha_i y_i \Phi\left(x_i\right)\Phi(x) + b$$

donc en fait on a juste besoin de connaître une fonction $K\left(x, x_i\right) = \langle x, x_i \rangle$! et c'est beaucoup plus simple.

Un exemple

$$\phi(x_1, x_2) = \left(1,\ \sqrt{2}x_1,\ \sqrt{2}x_2,\ x_1^2,\ \sqrt{2}x_1x_2,\ x_2^2\right)$$

$$\langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle = 1 \cdot 1 + \sqrt{2}x_1 \cdot \sqrt{2}y_1 + \sqrt{2}x_2 \cdot \sqrt{2}y_2 + x_1^2 \cdot y_1^2 + \sqrt{2}x_1x_2 \cdot \sqrt{2}y_1y_2 + x_2^2 \cdot y_2^2$$

$$\langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle = 1 + 2x_1y_1 + 2x_2y_2 + x_1^2y_1^2 + 2x_1x_2y_1y_2 + x_2^2y_2^2$$

$$\langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle = 1 + 2\mathbf{x}^T\mathbf{y} + (\mathbf{x}^T\mathbf{y})^2 = (1 + \mathbf{x}^T\mathbf{y})^2$$

On obtient une fonction beaucoup plus simple !

⚡ Limite des SVMs

- **Grande quantité de données** : la complexité d'entraînement est souvent quand même $O\left(n^2\right)$ ou $O\left(n^3\right)$ selon l'algorithme, ce qui peut devenir prohibitif si on a des centaines de milliers d'exemples.
- **Choix de kernel non trivial** : il n'existe pas de recette magique pour savoir quel kernel convient le mieux à un problème donné.