Introduction à l'apprentissage statistique - 2022

François Landes

September 8, 2023

Énoncés de TD et notes de cours éditées par François Landes [1]

Contents

[1] Avec l'aide de Kim Gerdes (l'autre prof de ce cours),
ainsi que l'aide initiale de quelques étudiants de la promo 2020-21: Alan Adamiak, Louise Allain, Mathieu Benard, Paul Michel Dit Ferrer, Ramdane Mouloua, Alexandre Pham

Pour tous les aspects organisationnels, se référer à "*cours0-orga.pdf*".

Bibliographie

- Site de bibliographies : http://lptms.u-psud.fr/francois-landes/enseignement/machine-learning-resources/
- *Hands On Machine Learning with Scikit Learn and TensorFlow*, **Aurélion Géron** (abordable, à la fois concret et assez complet) : https://github.com/yanshengjia/ml-road/tree/master/resources (en Français : *Introduction au Machine Learning*, **Aurélien Géron**)
- *Pattern Recognition and Machine Learning*, **Christopher Bishop**, 2006 (plus avancé, assez général)
- *Information Theory, Inference, and Learning Algorithms*, **Davis J.C. MacKay** (plus théorique, le mieux si vous aimez les probabilités)

References

Objectifs de l'enseignement

- Avoir des bases solides en apprentissage statistique, bien maîtriser les fondamentaux. C'est ce qui vous permettra d'approfondir.
- Découvrir le traitement automatique des langages (TAL ou NLP en anglais)
- Bien se repérer dans le vocabulaire du ML
- Avoir quelques réflexes de base, une connaissance basique du pipeline
- Plus concrètement, ce qui sera évalué :
  1. Connaître à fond quelques algorithmes (savoir en écrire le pseudo-code, les expliquer)
  2. À partir d'une description intuitive d'un algorithme, formaliser son expression mathématique, et logicielle. En lisant la documentation d'un algorithme, savoir le coder
  3. Pour un problème donné (tâche), trouver quelle classe de méthodes est pertinente
  4. Face à des situations classiques (résultats d'une expérience), analyser la situation de façon critique, et savoir faire les bons choix

Part I
Travaux Dirigés (TD) et Travaux Pratiques (TP)

Quand on parle d'exercices, c'est parce que littéralement, en les faisant, vous vous exercez à acquérir des automatismes en maths. C'est comme à la salle de gym: plus on muscle un muscle, et plus on parvient à soulever de lourdes charges sans trop d'effort. Bon entraînement !

Formules utiles:

$$\log(ab) = \log(a) + \log(b) \tag{1}$$

$$\frac{\partial}{\partial x}(u(x))^n = n(u(x))^{n-1}\frac{\partial}{\partial x}u(x) \qquad \text{ce qui s'écrit aussi:} \qquad (u(x)^n)' = nu'u^{n-1} \tag{2}$$

$$(u/v)' = \frac{u'v - uv'}{v^2} \tag{3}$$

$$(e^u)' = u'e^u \tag{4}$$

Si vous ne les connaissez pas, il faut les apprendre !! On ne vous les rappellera plus..

Lettres grecques

η: eta (minuscule)
θ, Θ: theta (minuscule, Majuscule)
ρ: rho
ϕ, φ, Φ: phi (minuscule, autre écriture minuscule, majuscule)
ϵ, ε: epsilon (minuscule, autre écriture minuscule)
ψ, Ψ: psi, Psi

Notations

Le produit scalaire entre deux vecteurs est noté de diverses façons: $\vec{a} \cdot \vec{b} = \mathbf{a}^T\mathbf{b} = \sum_i a_i b_i$
La p-norme d'un vecteur est définie comme :$\|\vec{a}\|_p = \left(\sum_i |a_i|^p\right)^{1/p}$
En particulier, la norme euclidienne correspond à:$\|\vec{a}\|_2 = \left(\sum_i |a_i|^2\right)^{1/2} = \sqrt{\sum_i a_i^2}$ Pour simplifier les notations, la norme euclidienne est souvent notée $\|\vec{a}\|$ au lieu de $\|\vec{a}\|_2$

1 Calculs de gradients (TD1)

1.1 Calcul de gradients – fonctions quelconques

Credit: Aurélien Decelle
Pour chacune des fonctions suivantes, calculez le gradient $\vec{\nabla} J$.

1. $J(\theta_1, \theta_2) = \theta_1^2 + \theta_2^2$
2. $J(\theta_1, \theta_2, \theta_3) = e^{-\theta_1^2}\theta_2 + \theta_3$
3. $J(\theta_1, \theta_2) = \frac{\theta_1^2}{\theta_2^2}$
4. $J(\vec{w}) = \vec{x} \cdot \vec{w}$, chaque vecteur a D éléments: on a $\vec{x} = (x_1, ..., x_D)$ et $\vec{w} = (w_1, ..., w_D)$. (Remarque: la notation $J(\vec{w})$ suggère de calculer $\vec{\nabla}_{\vec{w}} J$ et non pas $\vec{\nabla}_{\vec{x}} J$.)

[optionnel] $J(\vec{\theta}) = ||\vec{\theta}||_2 - a\ \vec{\theta} \cdot \vec{x}$, chaque vecteur a D éléments.

1.2 Calcul de gradients – fonctions bien choisies

Ici on a choisi des fonctions qui ont un intérêt en Apprentissage Statistique. Par ailleurs, on adopte les notations typiques du ML: $\vec{\theta}$ est le vecteur de paramètres, qui est donc la variable, alors que les données ($\vec{x}$) sont des valeurs fixées.

Pour chacune des fonctions $J(\vec{\theta})$ suivantes, calculez le gradient $\vec{\nabla}_{\vec{\theta}} J$. On a $\vec{\theta}$ un vecteur de dimension D.

1. $J_1(\vec{\theta}) = \frac{1}{2}||\vec{\theta}||^2 = \frac{1}{2}\vec{\theta}\cdot\vec{\theta}$
2. $J_2(\vec{\theta}) = \frac{1}{4}||\vec{\theta}||_4^4 = \frac{1}{4}(\theta_1^4 + \theta_2^4 + \ldots + \theta_D^4)$
3. $J_3(\vec{\theta}) = \frac{1}{2}\frac{1}{N}\sum_n^N (\vec{\theta}\cdot\vec{x}_n - y_n)^2$
4. $J_4(\vec{\theta}) = \frac{1}{4}\frac{1}{N}\sum_n^N (\vec{\theta}\cdot\vec{x}_n - y_n)^4$

[optionnel] $J_5(\vec{\theta}) = \|\vec{\theta}\|_1 = |\theta_1| + |\theta_2| + |\theta_3| + \ldots + |\theta_D|$ (là c'est la "valeur absolue" et pas la norme euclidienne).

[optionnel] Question nettement plus difficile ! (plus longue que l'ensemble des questions précédentes, vous pouvez y réfléchir un peu à la maison si ça vous intéresse): θ est maintenant une matrice de taille (K, D). On note $\vec{\theta}_k$ chacune de ses lignes (qui est donc un vecteur de taille D). On cherchera à dériver J_6 par rapport à un θ_k générique. $J_6 = \sum_\ell^K t_\ell \log(\hat{y}_\ell)$, avec $\hat{y}_\ell = \frac{e^{\vec{\theta}_\ell\cdot\vec{x}}}{\sum_k e^{\vec{\theta}_k\cdot\vec{x}}}$. Les indices ℓ et k vont de 1 à K.
Pour information, t_ℓ encode la vraie valeur à prédire, la "vérité terrain", c'est un vecteur "one-hot", c.a.d que une de ses composantes vaut 1, et les autres valent 0.
Cette fonction de perte est appelée *cross-entropy Loss*, elle intervient dans la classification multi-classe.

2 Descentes de gradient (TD2)

2.1 Descente de gradient – fonction d'une seule variable

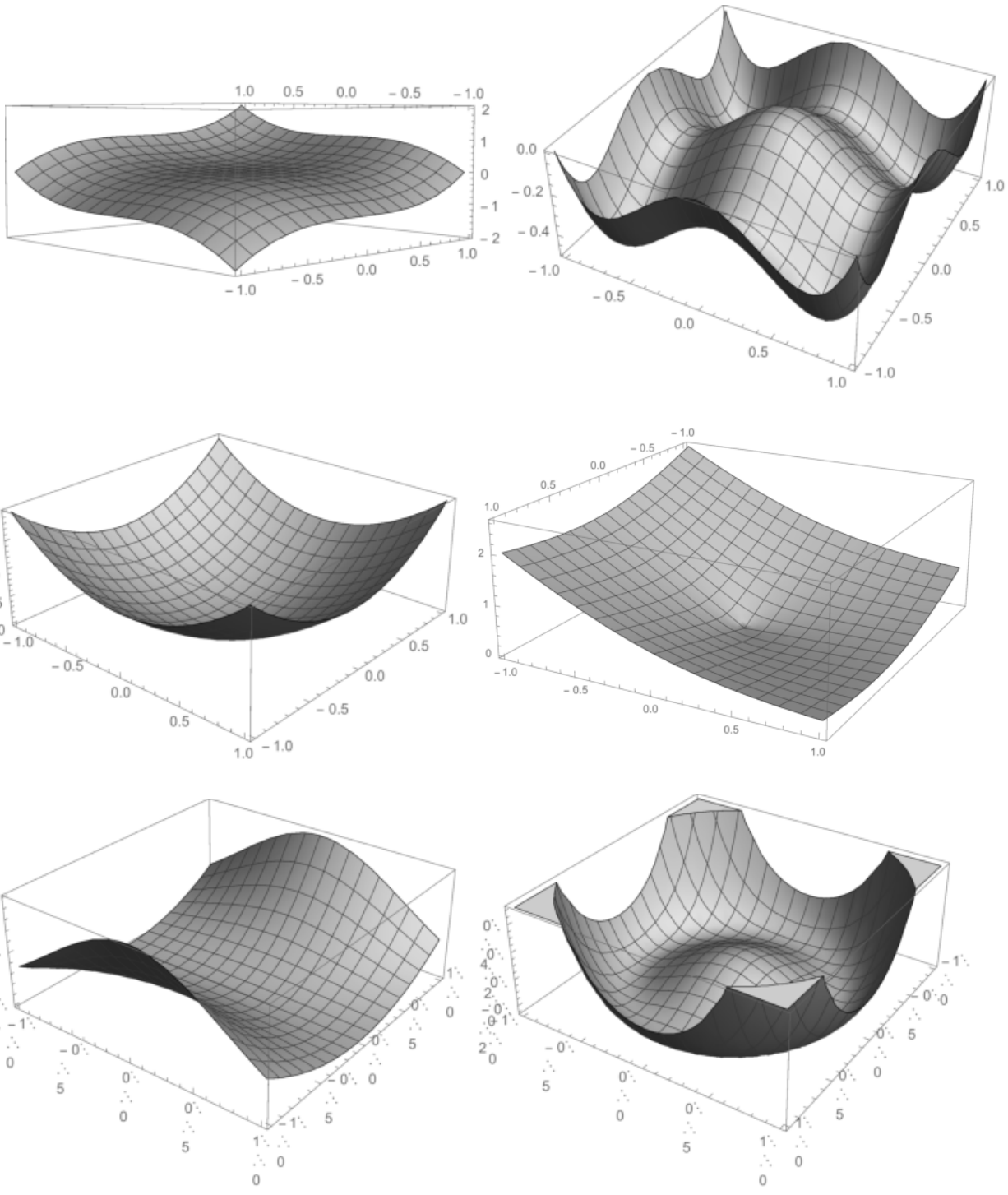
Crédit: Aurélien Decelle
Pour chacune des fonctions suivantes, dessiner schématiquement la fonction et indiquer le signe de la dérivée avec une flèche horizontale.

1. (a) $f_1(x) = 3x^2$, indiquer le sens de la dérivée aux points $x_1 = 1$ et $x_2 = -3$
   (b) $f_2(x) = x^3$, indiquer le sens de la dérivée aux points $x_1 = 0.1$ et $x_2 = -2$
   (c) $f_3(x) = -50x^2 + x^4$, indiquer le sens de la dérivée aux points $x_1 = -6$, $x_2 = -2$, $x_3 = 1$ et $x_4 = 10$
2. On va chercher un minium de f_1 par descente de gradient.
   (a) Expliquer comment se fait une mise à jour (détailler la formule générale dans le cas présent)
   (b) Expliquer ce qu'il se passe si l'on part de $x_0 = 1$ avec $\eta_1 = 1/3$ (faire 2 itérations).
   (c) Expliquer ce qu'il se passe si l'on part de $x_0 = 1$ avec $\eta_2 = 0.25 < 1/3$ (faire 2 itérations).
   (d) Expliquer ce qu'il se passe si l'on part de $x_0 = 1$ avec $\eta_3 = 0.5 > 1/3$ (faire 2 itérations).
3. On va chercher un minium de f_2 par descente de gradient.
   (a) Expliquer comment se fait la mise à jour (détailler la formule générale dans le cas présent)
   (b) Expliquer ce qu'il peut se passer si l'on part de $x_0 = 0.1$ avec un taux d'apprentissage petit (discutez sans forcément faire les calculs).
   (c) Expliquer ce qu'il peut se passer si l'on part de $x_0 = 0.1$ avec un taux d'apprentissage assez grand (discutez sans forcément faire les calculs).

2.2 Intuition sur des tracés de fonctions de deux variables

On donne ici le tracé de quelques fonctions, dont toutes ne sont pas mentionnées dans ce TD. Pouvez vous les reconnaître ? Aide: On a tracé les 6 surfaces suivantes:
$-(x^2+y^2)+(x^2+y^2)^2$;
$-(x^2+y^2)+(x^4+y^4)$;
x^2+y^2;
x^3+y^3;
$e^{-x^2}y^2$;
$||\vec{x}|| - a\vec{w}\cdot\vec{x} = (x^2+y^2)^{1/2} - aw_1x - aw_2y$, avec $a=3, w_1=0.1, w_2=0.3$

Dessinez la direction de la descente de gradient en quelques endroits, sur chaque figure. Si vous n'êtes pas à l'aise avec la 3D, vous pouvez tracer des plans de coupe.



2.3 Mini-TP – Descente de Gradient

Commencez par revoir la démo sur le gradient vue en cours (cf "Descente de Gradient - topo.ipynb"): `https://gitlab.inria.fr/flandes/ias/-/raw/master/seance1-Vocabulaire+Regression+DescenteGradient/DescenteDeGradier ipynb?inline=false` (cliquez pour télécharger directement)

Ensuite, téléchargez le jupyter notebook "2-mini-TP–DescenteGradient.ipynb", et suivez les instructions indiquées: `https://gitlab.inria.fr/flandes/ias/-/raw/master/TP-sujets/2-mini-TP-DescenteGradient.ipynb?inline=false` (cliquez pour télécharger directement)

Dans ce mini-TP, on cherche à expérimenter numériquement avec la descente de gradient, en 1 ou 2 dimensions (fonctions de 1 ou 2 variables), un peu comme ça a été vu en cours.

2.4 Descente de Gradient – fonction de deux variables (Bonus)

Crédit: inspiré d'un exercice d'Aurélien Decelle

Dans cet exercice un peu plus théorique, on va montrer (dans un exemple très simple) que **le gradient est perpendiculaire aux lignes de niveaux** (surfaces où la fonction est de valeur constante), c.a.d. que **le gradient indique la direction de plus grande pente**. C'est en fait un résultat général, valide pour toutes les fonctions (dérivables).

1. $f_1(x, y) = x^2 + y^2$
   (a) Calculer le gradient en un point générique (x_0, y_0).
   (b) Établir une équation paramétrique $(x(t), y(t))$ correspondant à la ligne de niveau $L(k)$, c.a.d. l'ensemble des points du plan: $L(k) : \{t \in \mathbb{R} / f_1(x(t), y(t)) = k\}$ où k est une constante positive. Pour vous aider, tracez-la d'abord dans le plan. On notera $\vec{L}_k(t) = (x(t), y(t))$ les coordonnées ainsi paramétrisées.
   (c) Dessinez un vecteur de coordonnées $(2x_0, 2y_0)$, en notant qu'il est proportionnel au vecteur donnant la position du point courant, (x_0, y_0). Que représente ce vecteur ? Dessinez le en partant du point (x_0, y_0). Prenez par exemple $(x_0, y_0) = (1, 1)$
   (d) Le vecteur tangent $\vec{T}$ à la ligne de niveau peut être obtenu en dérivant les coordonnées de la ligne de niveau $\vec{L}_k(t)$ par rapport au paramètre t. Calculez $\vec{T}$ dans le cas général, et dessinez $\vec{T}$, en un point (x_0, y_0).
   (e) Montrer que le gradient est perpendiculaire à la ligne de niveau au point (x_0, y_0). Note: ce résultat est très général.

Bonus $f_2(x, y) = -a(x^2 + y^2) + b(x^4 + y^4)$, avec $a, b > 0$.
   (a) Calculer le gradient au point (x_0, y_0).
   (b) Calculer la position des différents extremums (là où $\vec{\nabla} f = \vec{0}$). Lesquels sont des minimums ?
   (c) Dans un plan (x, y) dessiner la direction du gradient.
   (d) Expliquer où finirait une descente de gradient en fonction du point de départ (et en fonction du taux d'apprentissage η).

Bonus $f_3(x) = \sum_i \theta(x - \alpha_i)$, pour un ensemble de α_i quelconque.
Rappel La fonction de Heaviside est:
$\theta(x) = 0$ si $x < 0$
$\theta(x) = 1$ si $x > 0$
   (a) Calculer la dérivée par rapport à x de la fonction
   (b) Dessiner schématiquement la fonction ainsi que sa dérivée
   (c) Expliquer alors pourquoi : $\sum_i \theta(\vec{x_i} \cdot \vec{w} - \alpha)$ ne fait pas un bon choix de fonction à minimiser pour un perceptron (on minimise par rapport à $\vec{w}$ dans le cas du perceptron).

3 Géométrie: équation de droite, de plan, formules utiles (TD3)

3.1 Encodages d'une équation de droite (TD3)

Credit: Aurélien Decelle

Dans cet exercice, on vérifie/découvre la correspondance d'écritures entre l'équation de droite "habituelle" $y = ax + b$ et la notation $w_0 + w_1x_1 + w_2x_2 = 0$. On se place dans le plan (Ox_1x_2), où les coordonnées des points sont repérées par (x_1, x_2).

1. Cas particulier des droites passant par l'origine. On considère l'équation suivante : $w_1x_1 + w_2x_2 = 0$, où w_1 et w_2 sont les paramètres de la droite. Tracez les droites suivantes:
   (a) $w_1 = 1$ et $w_2 = 1$
   (b) $w_1 = -1$ et $w_2 = 1$
   (c) $w_1 = 0$ et $w_2 = 2$
   (d) $w_1 = 3$ et $w_2 = 0$
2. Expliciter dans le cas général ($w_0 + w_1x_1 + w_2x_2 = 0$), comment s'exprime la pente de la droite et les différents cas particuliers ($w_1 = 0$ et/ou $w_2 = 0$). Vous pouvez le faire en considérant 2 points appartenant à la droite, ou plus simplement par identification. Expliciter comment s'exprime l'ordonnée à l'origine.

3.2 Distance point-droite: quelques exemples (TD3)

Notation: Si $\vec{x}_A = (1, x_1, x_2)$, on notera $\vec{x}'_A = (x_1, x_2)$.
De même, $\vec{w} = (w_0, w_1, w_2)$ et $\vec{w}' = (w_1, w_2)$.
Remarque: les vecteurs avec un "prime" peuvent être dessinés dans le plan à 2D, alors que les vecteurs avec 3 composantes ne peuvent y être représentés.
Rappel: si deux vecteurs sont colinéaires, la valeur absolue de leur produit scalaire est le produit de leurs normes: $\vec{x}//\vec{y} \implies |\vec{x} \cdot \vec{y}| = ||\vec{x}||.||\vec{y}||$ (car l'angle est 0 et $\cos(0) = 1$)
Rappel: si deux vecteurs sont orthogonaux, leur produit scalaire est nul $\vec{x} \perp \vec{y} \implies \vec{x} \cdot \vec{y} = 0$

On se donne la droite (dans le plan, en 2D donc) définie par le vecteur (augmenté) $\vec{w} = (2\sqrt{2}, \sqrt{2}, \sqrt{2})$. Soit les points O, A, B, C, de coordonnées dans le plan $\vec{x}'_O = (0, 0), \vec{x}'_A = (2, 0), \vec{x}'_B = (-1, -1), \vec{x}'_C = (1, -1), \vec{x}'_D = (1, -3)$.

1. On va calculer les distances droite-point pour O, A, B, C, D. On *rappelle* (cf. l'exo 3.5) que la distance depuis la droite définie par $\vec{w}$ vers un point $\vec{x}$ (en coordonnées augmentées) est donnée par: $\frac{\vec{w} \cdot \vec{x}}{||\vec{w}'||}$.
   (a) Calculer la norme du vecteur directeur de la droite, le vecteur $\vec{w}'$, c.a.d. calculer $||\vec{w}'||$.
   (b) Définir les vecteurs augmentés $\vec{x}_O, \vec{x}_A, \vec{x}_B, \vec{x}_C, \vec{x}_D$, à partir des vecteurs à 2 composantes $\vec{x}'_O, \vec{x}'_A, \vec{x}'_B, \vec{x}'_C, \vec{x}'_D$
   (c) Calculer maintenant les distances droite-point pour O, A, B, C, D.
2. Représentation graphique. Idéalement, utiliser du papier à petit carreaux. On *rappelle* (cf. l'exo 3.5) que le vecteur $\vec{w}' = (w_1, w_2)$ est orthogonal à la droite définie par $\vec{w}$; et que la distance (signée) depuis la droite vers l'origine du repère $\vec{x}' = \vec{0}'$ est donnée par $+\frac{w_0}{||\vec{w}'||}$.
   (a) Placer les points O, A, B, C, D dans le plan.
   (b) Représenter, dans le plan, la direction indiquée par $\vec{w}'$.
   (c) Tracer la droite encodée par $\vec{w}$, en utilisant votre connaissance de la distance *signée* droite-O.
   (d) Vérifier que B, D appartiennent bien à la droite $\mathcal{D}$.
   (e) Vérifier que vos calculs de distance sont cohérents avec ce que vous voyez sur votre feuille quadrillée

3.3 Mini-TP distance point-droite (à travailler chez soi après le TD3)

On va implémenter les formules démontrées ci dessus en python. L'idée est de répondre aux 4 questions ci-dessous, mais il y a un énoncé plus guidé ici:
`https://gitlab.inria.fr/flandes/ias/-/raw/master/TP-sujets/Mini-TP-distance-point-droite.ipynb?inline=false`

1. Initialiser une droite de paramètres aléatoires (définissez une fonction).

2. Représentez cette droite dans le plan (définissez une fonction *display*), et aussi le vecteur $\vec{w}'$ associée à cette droite (il y est orthogonal, pour rappel).

3. Calculez la distance (signée) la droite et un point quelconque (définissez une fonction *distancePointDroite*, ou plutot peut-être *distanceDroitePoint*).

4. Affichez la distance (signée) à la droite avec un dégradé de couleurs.

3.4 Régression Linéaire (Bonus)

Cet exercice consiste essentiellement à vous faire re-dériver les mises à jour de la Régression Linéaire par descente de Gradient, puis à l'implémenter en python. L'énoncé est partiellement répété dans le sujet disponible en ligne:
https://gitlab.inria.fr/flandes/ias/-/raw/master/TP-sujets/2022-exo3.4-demo-RegressionLineaire-codeeA
ipynb?inline=false

Le cadre de la régression linéaire vous est probablement déjà bien connu dans le cas où $x \in \mathbb{R}$. On cherche à minimiser la fonction coût "erreur quadratique moyenne" (*Mean Squared Error*):

$$J(\theta, X) = \frac{1}{N}\sum_{n}^{N}(f_\theta(x) - y)^2 \tag{5}$$

$$\text{avec le modèle:} \qquad f_\theta(x) = \theta_0 + \theta_1.x \tag{6}$$

Les données sont donc l'ensemble des couples $(x_n, y_n), n \in [1, N]$.

1 Calculer $\vec{\nabla}_\theta J$. Éventuellement, calculez d'abord $\frac{\partial J}{\partial \theta_0}$ puis $\frac{\partial J}{\partial \theta_1}$.

2. En déduire la formule de mise à jour d'une descente de gradient avec taux d'apprentissage $\eta = 0.01$.

3. Écrire le pseudo code de l'algorithme de descente de gradient (initialisation, mises à jour, critère d'arrêt, calcul de l'erreur).

4. Généraliser au cas où $\vec{x} \in \mathbb{R}^D$ avec donc $f_\theta(\vec{x}) = \vec{\theta} \cdot \vec{x} + \theta_0$, où $\vec{\theta} = (\theta_1, \theta_2, \ldots, \theta_D)$. Normalement, les équations ci dessus ne changent quasiment pas (il y a juste quelques flêches de vecteur à rajouter). Écrivez le pseudo code, sans préciser les boucles courant sur D (au lieu de les expliciter, utiliser les fonctions accessibles dans numpy).

5. Testez votre pseudo-code en l'implémentant avec python (et numpy!)

6. Affichez la position de la droite qui fitte les points au fur et à mesure des itérations de la Descente de Gradient. Vous pouvez aussi calculez la valeur de la fonction coût à chaque itération, et afficher son évolution au cours des itérations.

3.5 Distance point-droite (Démonstration) ($\sim$ Bonus)

Credit: Aurélien Decelle

D'après l'exercice 3.1, on peut toujours décrire les droites du plan avec l'équation: $\vec{w} \cdot \vec{x} = 0$ où $\vec{w} = (w_0, w_1, w_2)$ et $\vec{x} = (1, x_1, x_2)$. On cherche maintenant à relier le vecteur $\vec{w}$ à des propriétés géométriques de la droite $\mathcal{D}(\vec{w})$.

Notation: Si $\vec{x}_A = (1, x_1, x_2)$, on notera $\vec{x}'_A = (x_1, x_2)$.
De même, $\vec{w} = (w_0, w_1, w_2)$ et $\vec{w}' = (w_1, w_2)$.
Remarque: les vecteurs avec un "prime" peuvent être dessinés dans le plan à 2D, alors que les vecteurs avec 3 composantes ne peuvent y être représentés.
Rappel: si deux vecteurs sont colinéaires, la valeur absolue de leur produit scalaire est le produit de leurs normes: $\vec{x}//\vec{y} \implies |\vec{x} \cdot \vec{y}| = ||\vec{x}||.||\vec{y}||$ (car l'angle est 0 et $\cos(0) = 1$)
Rappel: si deux vecteurs sont orthogonaux, leur produit scalaire est nul $\vec{x} \perp \vec{y} \implies \vec{x} \cdot \vec{y} = 0$

1. Considérer deux vecteurs $\vec{x}_A$ et $\vec{x}_B$ qui appartiennent à la droite définie par $\vec{w}$ (i.e. $\vec{w} \cdot \vec{x}_A = 0$ et $\vec{w} \cdot \vec{x}_B = 0$).

   (a) En utilisant le fait que par 2 points passe une et une seule droite, montrer que $\vec{x}'_A - \vec{x}'_B$ est colinéaire à la droite définie par $\vec{w}$.

   (b) Montrer ensuite que **le vecteur $\vec{w}' = (w_1, w_2)$ est orthogonal à la droite définie** par $\vec{w}$.

2. On considère maintenant un point $\vec{x}_\ell$ qui se situe sur la droite. On décompose ce point en une partie orthogonale à la droite et une partie colinéaire : $\vec{x}'_\ell = \vec{x}'_\perp + \vec{x}'_\parallel$.

(a) A quoi se réduit le produit scalaire entre $\vec{x}'_\ell$ et $\vec{w}'$?

(b) Montrer que $||\vec{x}'_\perp|| = \frac{|w_0|}{||\vec{w}'||}$.

(c) Montrer que **la distance (signée) depuis l'origine du repère, $\vec{x}' = \vec{0}'$ vers la droite** est donnée par $-\frac{w_0}{||\vec{w}'||}$. Astuce: considérer le projeté orthogonal P de l'origine O sur la droite, et remarquer que la longueur du vecteur OP définit la distance entre l'origine et la droite. Attention, cela signifie que **la distance (signée) depuis la droite vers le point origine, est, elle, de $+\frac{w_0}{||\vec{w}'||}$** (le signe change selon qu'on regarde la distance depuis la droite ou depuis le point).

3. (Difficile) [Cette question est difficile, on peut aussi admettre le résultat et avancer dans le TD] On considère maintenant un point quelconque dans l'espace : $\vec{x}$. Montrer que **la distance (signée) depuis la droite vers le point quelconque $\vec{x}$ est** $\frac{\vec{w}\cdot\vec{x}}{||\vec{w}'||}$. Indice: il peut être bon de considérer le projeté orthogonal, A, du point x sur la droite $\mathcal{D}(w)$.

4. (Bonus) Généraliser cette formule pour le cas d'un hyper-plan à D dimensions: quelle est la formule pour la distance entre un point quelconque dans l'espace et un hyper-plan caractérisé par le vecteur $\vec{w}$?

3.6 Distance point-plan (intuitions) (Bonus)

C'est un peu la même chose que l'exercice précédent, mais sans démo, de façon plutôt intuitive, par analogies, et avec d'autres notations (désolé..).

Soit un plan P (ou $\mathcal{P}$) d'équation $\vec{p}\cdot\vec{x} + c = 0$. On pourra noter $||\vec{p}||$ la norme de p, c.a.d $||\vec{p}|| = \sqrt{\vec{p}\cdot\vec{p}}$.

1. En commencant par le cas simple $c = 0$, et le sous-cas simple $\vec{p} = (0, 0, 2)$ (en 3 dimensions, pour se fixer les idées), donner une formule pour calculer le projeté $\vec{a}'$ d'un point $\vec{a}$ sur le plan P. Indice: commencer par voir quel est ce plan P, puis voir comment "hacker" les coordonnées de a pour les projeter sur ce plan.

2. Généraliser pour p quelconque (toujours avec $c = 0$, c.a.d. un plan qui passe par l'origine du repère). Vous pouvez imaginer un changement de coordonnées. Indice: il n'y a pas tellement de raisonnement à faire, juste se convaincre intuitivement (les dessins sont autorisés).

3. Généraliser pour c quelconque. On rappelle que $-c/||\vec{p}||$ s'interprète comme la distance signée entre le plan et l'origine du repère, c.a.d. la longueur du vecteur OO', si O est l'origine du repère et O' son projeté sur le plan.

Remarque: cette équation est valide en toute dimension $d \in \mathbb{N}$! Sympa, non ?

4 TP: Algorithme du Perceptron (TDs 3 et 4)

4.1 Perceptron – Pseudo code (TD3)

Écrire (rappeler) le pseudo code de l'algorithme du perceptron tel que vu en cours[2]. On précisera bien la taille des vecteurs et matrices utilisés, de cette façon: $X_{N,D}$ en indiquant le sens de chacune des variables. On tâchera de choisir des noms d'indices parlants (par exemple $d = 1, 2, \ldots, D$).

Idéalement, écrivez cet algo aussi sous forme numpy-optimale, c.a.d. en utilisant autant que possible des array, des masques (filtres booléens) plutôt que des boucles.

Bonus: écrivez aussi l'algo Online (d'un certain point de vue, c'est plus simple). Là, il n'y a pas beaucoup de possibilité d'écriture sous forme numpy-optimale.

4.2 Perceptron: papier-crayon (TD3-4)

Soient les données d'apprentissage suivantes, qu'on souhaite utiliser pour entraîner un classificateur binaire:

$$
\begin{array}{lll}
\text{Données 1 :} & X_1 & = \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 4 \\ 0 \end{pmatrix}, \begin{pmatrix} 4 \\ 1 \end{pmatrix} \right\} \\
& Y_1 & = \{-1, -1, 1, 1, 1, 1\} \\
\text{Données 2 :} & X_2 & = \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 4 \\ 0 \end{pmatrix}, \begin{pmatrix} 4 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0.5 \end{pmatrix} \right\} \\
& Y_2 & = \{-1, -1, 1, 1, 1, 1, 1\} \\
\text{Données 3 :} & X_3 & = \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right\} \\
& Y_3 & = \{-1, 1, 1, -1\}
\end{array}
$$

1. Visualiser les données: ici sans ordinateur mais pour des données peu nombreuses et de petite dimension (2), il suffit de les tracer dans le plan.
2. Lesquels de ces jeux de donnés sont linéairement séparables? Pourquoi ? Voyez vous des "remèdes" aux "problèmes" des données non linéairement séparables?
3. On considère l'algorithme *Full batch*. On détermine l'ensemble des points mal classés à chaque itération, et on modifie le vecteur des paramètres en sommant les contributions de tous les points mal classés. Exceptionnellement, pour simplifier les calculs à faire à la main, on considèrera la fonction coût sans le terme 1/N, c.a.d on prendra $J = \sum_{n=1}^{N} max(0, -(\vec{w} \cdot \vec{x}_n)t_n)$.
   (a) Faire "tourner" l'algorithme sur les données séparables (à la main d'abord). Dans chacun des cas, on choisira comme vecteur normal initial : $w = (1\ 0\ 0)^t$, et comme taux d'apprentissage, $\eta = 1$. En combien d'itération l'algorithme converge-t-il ?
   (b) Faites quelques itérations de chaque algorithme sur les données non séparables. Que se passe-t-il ?
4. Bonus: faire de même pour l'algo dans sa version *Online*.

4.3 Implémentation en Python (TD4)

Reprenez le fichier vu en cours, "2-exemple-Perceptron-qq-iterations.ipynb". Notez qu'il fait référence à un module "mes_fonctions", que vous n'avez pas (c'est fait exprès) ! Vous ne pouvez donc pas le faire tourner.

1. Complétez donc le fichier "TP4-mes_fonctions.py", dans lequel vous coderez l'algorithme du perceptron à la main (ce qui implique des gradients, des distance point-droite, et des mises à jour par descente de gradient).
2. Saisissez les données de l'exercice 4.2 dans votre version de "2-exemple-Perceptron-qq-iterations.ipynb", et faites tourner votre algo, vérifiez ce qui a été vu sur papier.
3. Également, partir d'un vecteur w initial aléatoire, observer les variations des résultats (w final et nombre d'itérations) selon la condition initiale.

[2] Décrivez l'algorithme standard, dit aussi *full Batch*, et non pas l'algo *Online*.

5 TP: Principal Component Analysis (Analyse en Composante Principale) (TD5)

Il y a différentes ressources sur le site du cours/le dossier partagé.
Le sujet peut se résumer ainsi:

1. Télécharger "TP-PCA+SVM-parties1-2-3-enonce.ipynb" : `https://gitlab.inria.fr/flandes/ias/-/raw/master/TP-sujets/TP-PCA+SVM-parties1-2-3-enonce.ipynb?inline=false`
2. Consulter la doc. de *sklearn.decomposition.PCA*.
3. Complétez en vous laissant glisser par les cases vides/les TODO... Mais les consignes ci dessous peuvent vous aider à comprendre la logique générale du TP.

5.1 PCA: prise en main

Ici, on souhaite faire une première PCA sur les données de MNIST (où n'importe quelles données visuelles).

1. Consultez la partie 1 du TP (fichier .ipynb)
2. Calculer la transformée en PCA des données en conservant 95% de la variance des données. Doit-on calculer la PCA seulement sur les données d'entraînement, ou bien sur l'ensemble (train+validation) ? Pourquoi ? (Quel est le risque, dans le mauvais cas ?)
3. Extraire le nombre de composantes effectivement retenues par la décomposition.
4. À partir de la version transformée des données, calculer aussi leur version décompressée (par transformée inverse). Afficher quelques exemples de avant/après (compression+décompression). On peut jouer avec le taux de variance expliquée et voir l'effet sur les images. (il est recommandé d'écrire une fonction).
5. Calculer l'erreur de reconstruciton moyenne sur toutes les images (c'est plus facile que pour une seule image). Le faire aussi pour l'ensemble de valdiation.
6. Calculer l'erreur de reconstruciton pour une seule image. (inclure ceci dans la fonction de comparaison avant/après).

Remarque: la variance expliquée ne mesure pas la même chose que l'erreur quadratique moyenne de reconstruction. En effet, la variance des données correspond à la dispersion des valeurs des pixels sur l'ensemble du dataset, alors que l'erreur de reconstruction mesure le taux de perte, pixel par pixel (et image par image), entre l'image de départ et reconstruite. Ce sont deux notions différentes.

5.2 Optimisation du nombre de composantes

1. Consultez la partie 2 du TP (fichier .ipynb)
2. En faisant varier explicitement le nombre de composantes (*nComp*) à retenir dans la PCA, observez comment les performances dépendent de ce nombre de composantes.
3. Sur le graphe, indiquer le point de meilleur score (le code vous est déjà donné, mais esayez de la faire par vous même/de bien comprendre le code).

5.3 Optimisation double: $nComp$ et C

1. Consultez la partie 3 du TP (fichier .ipynb)
2. Vous devez chercher à optimiser à la fois *nComp* et le paramètre C asocié à la réglarisation du SVM. Une grosse part du travail à déjà été fait. Prenez garde à appliquer la transformation PCA au bon moment, de la bonne façon.
3. Le tracé de la figure (score en fonction des deux hyper-paramètres) est déjà fourni. Lisez le attentivement (vous pouvez aussi l'améliorer, par exemple en faisant un tracé "en 3D").
4. En cherchant dans la doc de *np.argmax* et de *np.unravel*, obtenez les coordonnés du point qui réalise le maximum de score, et affichez le.
5. Refaire tout cela mais en utilisant par exemple le modèle des k plus proches voisins (k-NN, cf. la doc. de *sklearn.neighbors.KNeighborsClassifier*, en faisant varier le nombre de voisins au lieu de C).

5.4 Question subsidiaire

1. Si ce n'est pas déjà fait, factorisez votre code en 1 seul fichier, avec les trois parties ci dessus mises dans des blocs de sorte à être facilement testés indépendamment (par exemple, avec trois blocs *if*).

2. Testez votre code sur d'autres jeux de données: par exemple le MNIST complet (en résolution 28x28), ou Fashion-MNIST. Attention, pour les gros jeux de données, il est recommandé de restreindre les plages de paramètres explorés, ou alors de restreindre le nombre d'exemples d'entraînement et de validation.

5.5 Bonus – PCA et reconstruction

1. Écrire le pseudo code détaillé d'une analyse en composantes principales. (En supposant que l'opération de diagonalisation est effectuée par l'appel à une librairie d'algèbre linéaire).
2. Écrire aussi la transformée inverse (décompression), mathématiquement, et en pseudo-code.
3. Vérifiez que votre code est cohérent avec le résultat produit par la librairie *sklearn*.

6 Analyse de résultats d'expérience (TD6)

Ces exercices visent à vous apprendre à savoir rapidement reconnaître quelques situations classiques, et plus généralement, à savoir analyser des comportements erratiques lorsqu'on fait de l'apprentissage statistique.

Dans les figures ci dessous, on a toujours :

- En ordonnées la *accuracy* (une mesure de performance)
- En abscisse un hyper-paramètre, parfois le nombre d'époques (nombre d'itérations, parfois le nombre de features, parfois autre chose).

Parfois, toutes les informations ne sont pas fournies, et c'est à vous de "deviner" certaines choses.

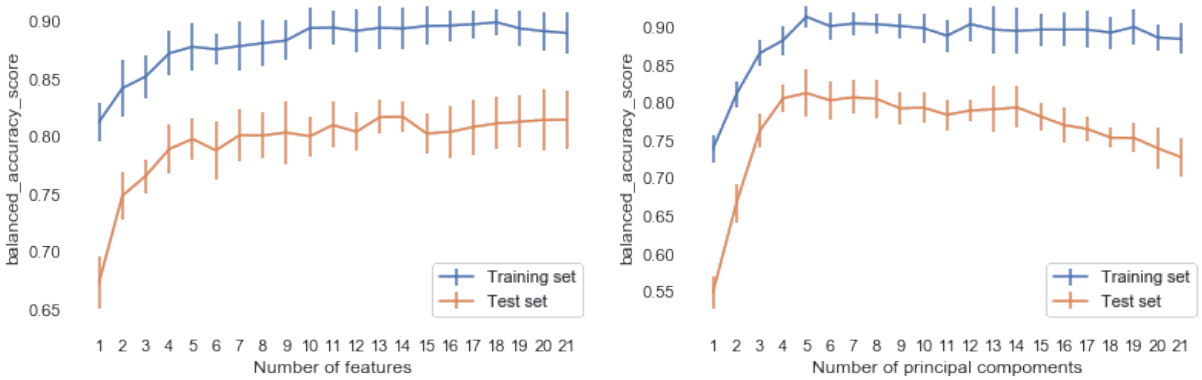


Figure 1: Ici l'hyper-paramètre en abscisse est le nombre de features conservées pour effectuer une classification. On peut supposer qu'une technique de réduction dimensionnelle intervient en amont. Questions:
1.1 Comment interpréter les intervalles d'incertitude qui sont affichés ? À votre avis, comment ont-ils été calculés ?
1.2 Pour chacune de ces deux expériences, que concluez vous ? En particulier, quelle valeur de l'hyper-paramètre (abscisse) choisiriez vous ? Justifiez bien la réponse (le raisonnement compte plus que le choix en lui même, plusieurs réponses sont admissibles).
1.2 À quel point votre choix est il fait avec certitude ?
Source: perdue :(

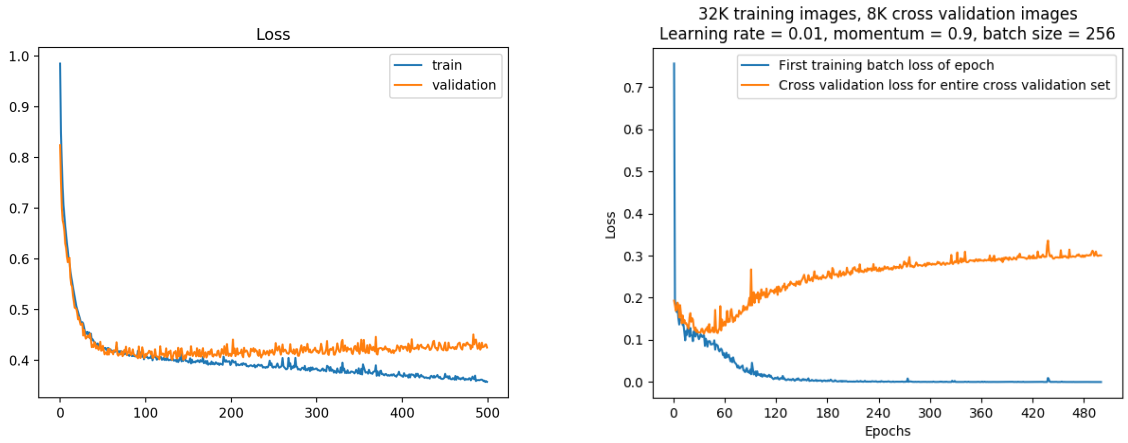


Figure 2: Ici l'axe des abscisse indique le nombre d'époques effectuées. La *Loss* est la fonction de *perte*, c'est-à-dire la fonction coût: plus elle est basse, mieux c'est. Questions:
2.1 Dans les deux cas ci dessus, à quel instant de l'apprentissage (quelle epoch) vous arrêteriez vous?
2.2 Pouvez vous dire lequel de ces modèles est le plus en situation de sur-paramétrisation ? Dans quelle mesure la comparaison des graphes ci-dessus est elle trompeuse ?
Gauche: Figure issue de `https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-mo`
Remarque: Cette page web contient des imprécisions, mais les figures me sont utiles comme illustrations simple (même si il y manque les légendes des axes, etc..). Bref je ne recommande pas.
Droite: Figure issue de `https://towardsdatascience.com/my-first-kaggle-competition-9d56d4773607`
(Dont les conclusions, par exemples, sont intelligentes - je n'ai pas lu toute la page, donc je ne garanti rien).

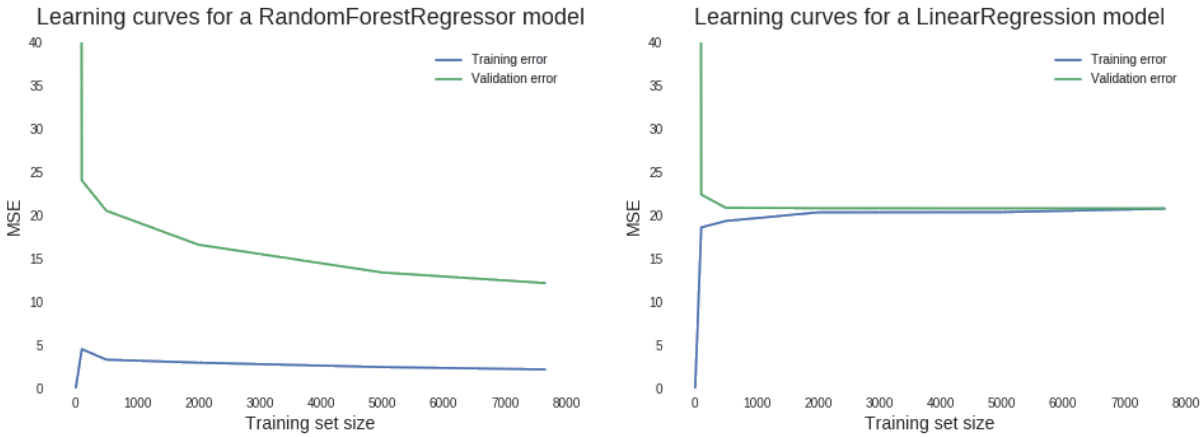


Figure 3: Ici, l'abscisse n'est pas vraiment un hyper-paramètre, mais la taille du *training set*, c'est-à-dire le nombre de points de données utilisés pour la phase d'entraînement. Ce genre de courbe est appelée *learning curve* ou *courbe d'apprentissage* (comme lorsqu'un humain apprend quelque chose, comme un nouveau langage de programmation: avec le temps, on progresse, plus ou moins vite selon l'avancement de l'apprentissage).
Questions:
2.1 Pour les deux modèles ci dessus, lequel choisiriez vous ?
2.2 Si on vous propose d'agrandir la taille de l'ensemble d'apprentissage, êtes vous intéressé(e) ?
2.3 Lequel des modèles semble être sous-paramétrisé (en situation d'under-fitting) ?
`https://www.dataquest.io/blog/learning-curves-machine-learning/`

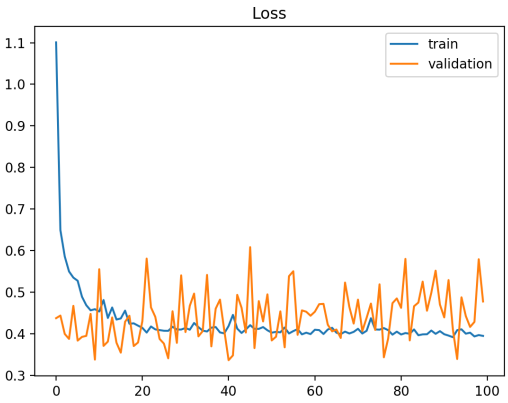


Figure 4: Ici encore, figure baclée: l'axe des abscisse est le nombre d'époques, et l'axe des ordonnées est la *Loss*. Questions:
3.1 Ci-contre, on a l'erreur (train et validation) en fonction de l'époque. Au vu des fluctuations importantes de l'erreur de validation, que peut on dire sur la taille du validation set ? On peut aussi tenter de discuter la taille de training set.
Figure issue de `https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-mo`
(Même remarque que précédemment, page de qualité très moyenne, mais on cite ses sources !).

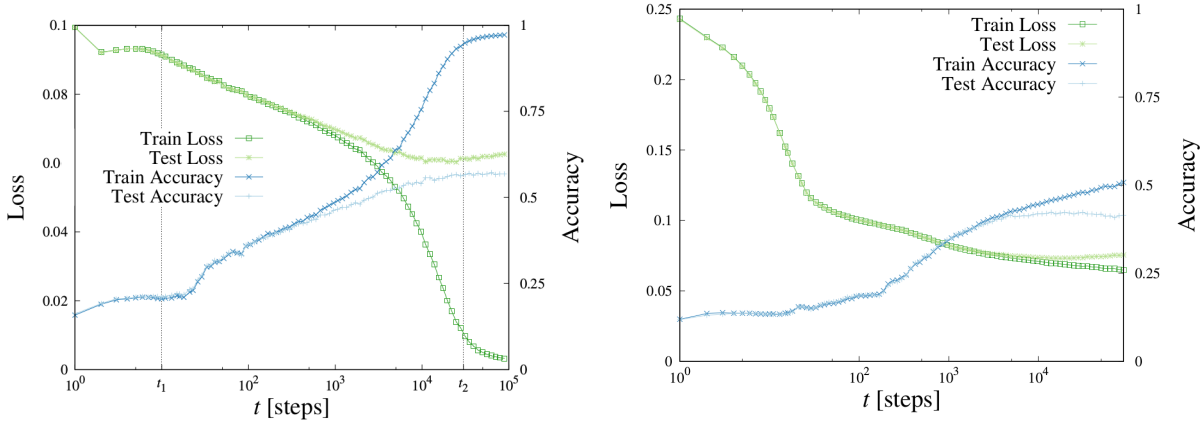


Figure 5: Parmi les deux modèles ci dessus, l'un est sous-paramétrisé (modèle trop peu expressif, résultant en de l'under-fitting), et l'autre est sur-paramétrisé (modèle trop expresif, résultant en de l'over-fitting). L'axe des ordonnées est double: à gauche, la *Loss*, à droite, la *accuracy* (ceci, dans chacune des deux figures). Questions:
5.1 Faites l'association, en justifiant votre réponse.
5.2.a En supposant qu'on vous présente chacun de ces modèles séparément, et que vous ne pouvez en modifier l'expressivité, mais que vous pouvez choisir d'interrpompre l'entraînement à l'instant t (steps) de votre choix, quel instant choisiriez vous dans le premier cas ?
5.2.b Dans le deuxième cas ?
5.3 Avec ces choix de temps d'arrêt, lequel des deux modèles choisiriez vous ?
Figures issues de l'article [**?**].

7 TP: coder un algo simple et analyser le résultat (TD7)

TP à définir. Ou alors, utiliser ce temps pour approfondir le TD5, avoir du temps de marge lors du TP4
Cette séance accueillera p-e le TP noté !

8 Exercices de type "prise en main d'un problème" (TD8)

Cet exercice est un **bon échauffement pour savoir démarrer votre projet**. Il y aura aussi un exo de ce type (parmi 3-4 exos) à l'examen.

Dans cet exercice, nous vous demandons de donner une idée générale du problème, et non de préciser tous les détails sous forme mathématique. Vous pouvez utiliser des mots et évenutellement des croquis simples pour expliquer vos idées. L'idée est de **concevoir la feuille de route pour résoudre un problème**, sans forcément le faire effectivement.

Dans chaque cas, vous avez **un notebook qui vous montre commment réaliser un chargement sommaire** des données: `https://gitlab.inria.fr/flandes/ias/-/raw/master/TP-sujets/Analyse-de-tache.zip?inline=false`

Le premier exercice est déjà corrigé. Nous discuterons la correction en TD.

Pour chacun des projets suivants, répondez à ces questions :

(a) Quel est l'objectif, globalement (en français) ?

(b) Quel type de tâche devrions-nous probablement accomplir (supervisée ou non, et dans chaque cas, quelle sous-catégorie) ?

(c) Quelle est la structure des données ? (vous pouvez et devriez souvent faire certaines suppositions, selon les besoins). Faut-il faire attention à certains aspects du formattage ?

(d) Quel algorithme semble bien adapté ?

(e) Y a-t-il des points de vigilance particuliers ? (choix des hyperparamètres, mesure des performances, ensemble de données non équilibrées, etc).

Hormis les points (c),(e), les questions reçoivent une réponse sur une seule ligne ! Dans certains cas il y a une question supplémentaire (e-bis) qui complète la question (e). Pour la question (e), à chaque fois, quelques lignes (environ 5) devraient suffire pour la discussion. Soyez concis.

Sources:
`https://www.kaggle.com/uciml/mushroom-classification`
`https://www.kaggle.com/antoinekrajnc/soccer-players-statistics`
et `https://www.kaggle.com/hugomathien/soccer`
`https://www.kaggle.com/c/whats-cooking`
`https://www.kaggle.com/terminus7/pokemon-challenge`

8.1 La cueillette des Champignons

Nous avons un ensemble de données sur les propriétés des champignons, dont un attribut binaire, "comestible/empoisonné". Les autres attributs sont souvent catégoriques : forme du chapeau, surface du chapeau, couleur du chapeau, odeur, habitat, etc. Il existe beaucoup plus de champignons comestibles que de champignons vénéneux. Nous sommes **intéressés à ne pas mourir du poison**.

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-size	gill-color	stalk-shape	stalk-root
0	p	x	s	n	t	p	n	k	e	e
1	e	x	s	y	t	a	b	k	e	c
2	e	b	s	w	t	l	b	n	e	c
3	p	x	y	w	t	p	n	n	e	e
4	e	x	s	g	f	n	b	k	t	e

Voir le fichier '*mushrooms-chargement-minimal.ipynb*'.

Ne sautez pas sur la correction ! Essayer de réfléchir un peu avant de lire la suite !

Correction (a) Le but est d'identifier si un champignon est comestible ou pas, à partir de certaines informations (features ou attributs) données sur ces champignons. On souhaite que l'algorithme fonctionne y compris sur des champignons dont l'espèce n'était pas présente dans le training set.

Correction (b) C'est une tâche de classification binaire, car on a que deux classes ici (contrairement à de la classification multi-classe par exemple).

Correction (c) Tout les attributs sont catégoriels. Il faudra penser à les encoder sous forme de one-hot vectors. Comme tous sont catégoriels, c'est pratique, on pourra appliquer les mêmes types de pré-traitement à tous les attributs.

Correction (d) La plupart des algorithmes de classification peuvent être utilisés ici. Toutefois, les attributs étant catégoriels, les algorithmes basés sur des arbres de décision peuvent être efficaces, a priori (ou par dessus, des méthodes ensemblistes avec vote.. on en parlera plus tard).

Correction (e) Il est dit dans l'énoncé que les classes ne sont pas équilibrées. Il faudra donc veiller à ce que la classe "poisonous" soit suffisamment représentée ! En fait, ici, le "coût" d'un faux négatif (prédire que c'est comestible alors que ça ne l'est pas) est immensément plus "cher" (mourrir?) que le coût d'un faux positif (se priver d'un bon petit champignon commestible, par erreur).

8.2 Combats de Pokemon

Dans le jeu de pokémon, chaque pokémon a des capacités particulières, qui peuvent être très efficaces contre un autre type de pokémon. Un combat est effectué avec un ensemble fixe de pokémons de chaque côté du combat. Nous avons ici un cas simple où il n'y a qu'un seul pokémon de chaque côté. Nous avons un ensemble de données de milliers de combats de pokémon (50 000), pour un ensemble de 800 pokemons de caractéristiques fixées:

Combats:

	First_pokemon	Second_pokemon	Winner
0	266	298	298
1	702	701	701
2	191	668	668
3	237	683	683
4	151	231	151
...	...	...	...
49995	707	126	707
49996	589	664	589
49997	303	368	368
49998	109	89	109
49999	9	73	9

Pokedex:

	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Gen	Legendary
0	Bulbasaur	Grass	Poison	45	49	49	65	65	45	1	False
1	Ivysaur	Grass	Poison	60	62	63	80	80	60	1	False
2	Venusaur	Grass	Poison	80	82	83	100	100	80	1	False
3	Mega V	Grass	Poison	80	100	123	122	120	80	1	False
4	Charmander	Fire	NaN	39	52	43	60	50	65	1	False
...	...	...	...	...	...	...	...	...	...	...	...
795	Diancie	Rock	Fairy	50	100	150	100	150	50	6	True
796	M Diancie	Rock	Fairy	50	160	110	160	110	110	6	True
797	Hoopa C	Psychic	Ghost	80	110	60	150	130	70	6	True
798	Hoopa U	Psychic	Dark	80	160	60	170	130	80	6	True
799	Volcanion	Fire	Water	80	110	120	130	90	70	6	True

Dans chaque cas, nous connaissons certaines caractéristiques de chaque pokémon : Nom, points de vie, attaque, défense, vitesse, type (eau, feu, psy, terre, etc.). Le combat est toujours gagné par une équipe, l'équipe

1 ou l'équipe 2. Nous sommes **intéressés à prédire qui peut gagner un combat donné**.

Après avoir réfléchi par vous-même, vous pouvez aller voir le fichier '*pokemons-combats-chargement-minimal.ipynb*'.

8.3 Positions des footballeurs

Dans cet exercice, nous vous demandons de donner une idée générale du problème, et non de préciser tous les détails sous forme mathématique. Vous pouvez utiliser des mots et éventuellement des croquis simples pour expliquer vos idées. L'idée est de **concevoir la feuille de route pour résoudre un problème**, sans le faire effectivement.

On aimerait prédire la position de jeu d'un footballeur en fonction de différentes informations sur ce joueur. L'idée est par exemple de suggérer des positions de jeux pour des joueurs dont on connaît les caractéristiques, mais pas la position de jeu. On dispose de données sur 17588 de joueurs professionnels. Voici un extrait du jeu de données:

Nom	Posi club	Club	Taille	Poids	Pied	Age	Controle Balle	Interce -ption	Passe courte	Passe longue
Ronaldo	ST	Real M..	185 cm	80 kg	Right	32	93	29	83	77
Messi	RW	FC Bar..	170 cm	72 kg	Left	29	95	22	88	87
Neymar	LW	FC Bar..	174 cm	68 kg	Right	25	95	36	81	75
Suárez	ST	FC Bar..	182 cm	85 kg	Right	30	91	41	83	64
Neuer	GK	FC Bay..	193 cm	92 kg	Right	31	48	30	55	59
De Gea	GK	Manche..	193 cm	82 kg	Right	26	31	30	31	32
Lewand.	ST	FC Bay..	185 cm	79 kg	Right	28	87	39	83	65
Bale	RW	Real M..	183 cm	74 kg	Left	27	88	59	86	80
Ibrahimo.	nan	Manche..	195 cm	95 kg	Right	35	90	20	84	76
Courtois	GK	Chelse..	199 cm	91 kg	Left	24	23	15	32	31

La colonne **Nom** est le nom du joueur (17588 joueurs différents) (on a caché le prénom, mais on l'a aussi, mais ça ne rentrait pas dans une feuille A4 pour l'examen).
La colonne **Position club** prend comme valeurs 'ST' 'RW' 'LW' 'GK' 'RW' nan 'RCB' ... (12 valeurs différentes)
La colonne **Club** prend comme valeurs les noms des clubs (634 clubs différents).
La colonne **Taille** prend des valeurs comme '155 cm', '207 cm', '181 cm' (50 valeurs différentes)
La colonne **Poids** prend comme valeurs '49 kg', '90 kg' (56 valeurs différentes)
La colonne **Pied** prend comme valeurs 'Right' 'Left' 'Both'
La colonne **Age** prend des valeurs entières de 17 à 47
La colonne **Controle Balle** prend des valeurs de 5 à 95
La colonne **Interception** prend des valeurs de 3 à 93
La colonne **Passe court** prend des valeurs de 10 à 92
La colonne **Passe longue** prend des valeurs de 7 à 93

Les questions (a),(c),(d),(e) reçoivent une réponse courte !

a. (0.5 pt) Quel est l'objectif du projet ? (en français, pas en langage mathématique). Quel type de tâche devons-nous donc accomplir (supervisée ou non, et dans quelle sous-catégorie se trouve-t-on: classification ou régression, clustering ou estimation de densité, etc) ?

b. (0.5 pts) Y a t il besoin de nettoyer les donner, de convertir/remplacer certaines entrées ?

c. (2 pts) Comment formateriez vous les données (encodage des données) ? Répondez colonne par colonne. Faut-il faire attention à certains points en particulier ?

d. (0.5 pts) Expliquer pourquoi il vous semble pertinent de supprimer certaines colonnes (2 peuvent légitimement être supprimées: à vous de voir lesquelles et expliquer *pourquoi*). Ne vous appuyez pas sur vos connaissances en football pour choisir quoi supprimer, mais uniquement sur des arguments mathématiques, concernant la quantité d'information relative aux colonnes.

e. (0.5 pts) Quels sont les étapes de pre-processing qui vous semblent nécessaires/souhaitables, en dehors de l'encodage que vous venez de décrire ?

f. (0.5 pts) Quel algorithme vous semble bien adapté, parmi SVC, SVR, Decision Trees, Regression Trees, Régression linéaire (avec, sans noyau), Perceptron ? Jjustifiez brièvement.

g. (0.75 pts) Faut il prendre garde à l'équilibre du dataset ? Quel serait le déséquilibre potentiel à prendre en compte, ici ?

h. (0.75 pts) Y a-t-il des points de vigilance particuliers concernant la mesure des performances ? Quelle quantité serait il bon d'afficher, pour bien contrôler les performances ?

i. (Bonus, nécessitant des connaissances hors-programme) Il y a un problème supplémentaire à cette tâche. Les étiquettes des positions de jeu ne sont pas normalisées dans le monde entier. Au départ, nous avons en fait non pas 12 mais 27 valeurs possibles pour les positions de jeu dans notre ensemble de données, ce qui est beaucoup plus que les environ 11-12 positions qui sont généralement considérées par les experts en football. Nous souhaitons réduire cette diversité de positions de jeu à un nombre raisonnable (par exemple, de 27 vers 12). Comment procéderiez-vous pour effectuer cette réduction automatiquement (à partir des données, pas à partir de votre expertise footballistique), qui est un préalable à la tâche décrite ci-dessus ?

8.4 Cuisines du monde (recettes)

Après avoir réfléchi par vous-même, vous pouvez aller voir le fichier '*whats-cooking-chargement-minimal.ipynb*'.

On dispose d'un jeu de données de recettes de cuisine de différents pays. Pour chaque recette, on a d'une part le pays d'origine, et d'autre part tous les ingrédients apparaissant dans la recette. On peut envisager de voir si la nationalité d'origine d'une recette est prédictible à partir des seuls ingrédients de la recette.

	id	cuisine	ingredients
0	10259	greek	[romaine lettuce, black olives, grape tomatoes...
1	25693	southern_us	[plain flour, ground pepper, salt, tomatoes, g...
2	20130	filipino	[eggs, pepper, salt, mayonaise, cooking oil, g...
3	22213	indian	[water, vegetable oil, wheat, salt]
4	13162	indian	[black pepper, shallots, cornflour, cayenne pe...

8.5 Cas pratiques: quel pipeline construire ?

Pour chacune des tâches suivantes, (1) préciser rapidement le format des données, (2) déterminer le type de tâche, (3) proposer un algorithme parmi ceux vus en cours, (4) proposer un pre-processing, éventuellement, des données, (5) proposer un critère de performance. (6) éventuellement, faire la critique des limites de l'approche proposée et suggérer d'autres algorithmes, parmi ceux qui n'ont *pas* été vus en cours.

a) **Réseau social :** Pour aider les personnes dépassées par l'augmentation du nombre de leurs amis, un réseau social permet de créer automatiquement des groupes "intelligents" d'amis (leur nombre est paramétrable) en fonction du nombre de messages échangés avec chaque ami, la date des premiers et derniers contacts, le nombre d'amis communs...

b) **Gestion de photos :** Pour permettre aux utilisateurs de mieux gérer leurs photos et retrouver facilement leurs clichés, le logiciel de gestion des photos doit déterminer automatiquement si les photos sont des portraits de personnes ou des paysages de mer, de neige ou de forêt.

c) **Recommandation musicale :** Un service de musique en ligne enregistre les choix musicaux de ses clients (genre de musique, artistes, rythme, tonalité...) et souhaite déterminer des groupes de personnes qui partagent les mêmes goûts musicaux afin de leur recommander des nouveautés qui pourraient les intéresser.

d) **Traitement de déchets :** Vous êtes chargé de mettre en place un système de tri automatique sur une chaîne de traitement de déchets. Les déchets passent un par un devant une caméra qui enregistre une image en couleur de dimension 256×256. Votre système doit distinguer quatre types de déchets : des bouteilles en verre, des journaux en noir et blanc, des magazines en couleur et des petits pots de yaourt.

9 Exercices autour du Max. de Vraisemblance (MLE) (TD9-10)

9.1 Loi exponentielle - Estimation du Maxmimum de Vraisemblance (MLE en anglais)

Soit X une variable aléatoire réelle dont on suppose qu'elle suit la loi exponentielle: $\rho(X = x) = \lambda exp(-\lambda x)$. $\rho(x)$ dénote la densité, définie par l'évenement: $P(X \in [x, x + dx]) = \rho(x)\mathrm{d}x$. On dispose de N points de données, $x_1, x_2, \ldots, x_N$. Quel est l'estimateur du paramètre λ qui maximise la vraisemblance des données ?

9.2 TP: Algorithme Bayésien Naïf (TP9-10)

A. Téléchargez le fichier "8-TP-BayesienNaifSurMNIST.ipynb"

B. Si votre installation de sklearn est défectueuse ou n'est pas encore faite, NE passez PAS de temps à l'installer maintenant. Téléchargez plutôt "ManualLoad.npz".

C. Complétez le programme, en vous aidant des endroits indiqués "TODO". En particulier:

1. Si vous optez pour un modèle de Bernoulli, transformez les données de sorte à ce qu'elles vivent dans l'espace où une variable de Bernoulli est définie.
2. Définissez l'ensemble d'apprentissage et l'ensemble de test (cette fois ci il n'y a pratiquement pas d'hyper-paramètres, donc on ne fait pas forcément de validation).
3. Complétez les fonctions d'entraînement et de prédiction (si vous avez le pseudo code sous les yeux, ce devrait être très facile). Consultez les équations du poly de cours (fin du paragrpahe "Apprentissage des paramètres du modèle" et fin du paragraphe "Fonction de décision"). Pour rappel, les deux formules cruciales ici sont :

$$p_{k,d} = \frac{\sum_n^N x_{n,d} \mathbb{1}_{y_n=k}}{\sum_n^N \mathbb{1}_{y_n=k}} \tag{7}$$

$$P_k = \frac{\sum_n^N \mathbb{1}_{y_n=k}}{N} \tag{8}$$

$$k^* = \mathrm{argmax}_k \left(\sum_{d=1}^{D} \left[(x_d) \log(p_{k,d} + \varepsilon) + (1 - x_d) \log(1 - p_{k,d} + \varepsilon) \right] + \log P_k \right) \tag{9}$$

   Avec $\varepsilon = 10^{-8}$ par exemple.
4. Entraînez et testez votre modèle
5. Affichez les parametres appris sous un format lisible par l'humain.
6. Bonus: Une fois que tout fonctionne bien, vous pouvez éventuellement charger le jeu de données MNIST complet, c.a.d. en résolution 28×28 (contenu dans le fichier "mnist70.npz", il contient 70000 images).

D. En quoi le jeu de données MNIST est il critiquable (si on le compare à des jeux de données comme CIFAR-10, CIFAR-100 ou ImageNet) ? Que peut-on en déduire sur la performance attendue de ce genre de modèles, pour des images du genre CIFAR-10, etc ?

E. Bonus: appliquez cet algorithme à des données du même genre que dans les premiers TPs (utilisez le code fourni dans "exo1.4-générateur-de-blobs.py"). Il est alors recommandé d'utiliser le modèle Gaussien. Représentez les paramètres appris du mieux que vous pouvez. Il est aussi possible de générer de grandes quantités de données, puis de tracer la courbe d'apprentissage (*learning curve*), c.a.d. le taux d'erreur de test en fonction de la taille du *train set*, pour une taille de *test set* fixée.

9.3 Deux dés truqués - Estimation du Maxmimum de Vraisemblance (MLE en anglais) (Bonus)

Soit la loi du lancé de deux dés, qui sont peut être truqués: $P(X_1 = d_1) = p_1(d_1)$ pour le premier dé (rouge), et $P(X_2 = d_2) = p_2(d_2)$ pour le 2ème dé (bleu). On ne connait pas les fonctions p_1 et p_2 (qui

sont chacune des fonctions de $\{1,2,3,4,5,6\}$ dans $[0,1]$). Ces fonctions peuvent être paramétrés par 12 nombres $p_{1,1}, p_{1,2}, .., p_{1,6}, p_{2,1}, p_{2,2}, \dots p_{2,6}$.

1. En supposant que les deux dés sont indépendants, quelle est la loi de $X = (X_1, X_2)$? (En restant abstrait - il n'y a pas de piège !).

2. On dispose de la donnée de N lancés des deux dés qui ont patiemment été effectués: $(d_1, d_2)_1, (d_1, d_2)_2, (d_1, d_2)_3, \dots, (d_1, d_2)_N$. Sans faire de calcul, à votre avis, quel est l'estimateur du maximum de vraisemblance de la loi de X ?

9.4 Deux pièces truquées - Estimation du Maxmimum de Vraisemblance (MLE en anglais) (Bonus)

Supposons qu'on a deux pièces magiquement corrélées: lorsque la première tombe sur pile, la deuxième à aussi plus tendance à tomber sur pile (et vice-versa) (mais pas non plus à 100% hein, c'est juste plus corrélé que 2 pièces indépendantes). Le résultat du lancé des deux pièces (la première, et la seconde, qui sont de couleurs différentes) est noté PF, FP, etc Soit la variable aléatoire associée au lancé de ces deux pièces, ainsi définie: $X(PP) = (1,1), X(PF) = (1,0), X(FP) = (0,1), X(FF) = (0,0)$.

1. Peut on écrire simplement la loi de ces deux lancés? De combien de paramètres a-t-on besoin au minimum pour caractériser cette loi (indice: dénombrez toutes les issues possible du double lancé).

2. On dispose de la donnée de N lancés des deux pièces qui ont patiemment été effectués: $(d_1, d_2)_1, (d_1, d_2)_2, (d_1, d_2)_3, \dots, (d_1, d_2)_N$. Quel est le meilleur estimateur des paramètres de la loi des deux lancés ?

3. Un scientifique doute du caractère magiquement relié des deux pièces. Quelle quantité peut-on calculer, à partir des données, pour mesurer le degré de corrélation des deux pièces ?

4. Calculez cette quantité dans trois cas limites:

   - le cas où les 2 pièces sont en fait indépendantes,
   - le cas où les deux pièces sont toujours exactement du même coté (on n'a que des pile-pile ou des face-face dans TOUS les lancés),
   - et le cas où elles sont toujours exactement de coté opposé.

   Pour cette dernière question, si on est fatigués, après avoir un peu réfléchi aux calculs et essayé de "deviner" la solution, on peut le faire numériquement, dans un petit jupyter-notebook. (il existe une fonction np.cov qui calcule quelque chose d'intéressant.. elle est très facile à re-coder, avec la fonction np.dot, à la main !).

10 Autres séances de TD TP

Il y aura une séance dédiée au TP noté, et 2 séances dédiées à des TP de TAL, mais qui n'ont pas de support correspondant dans le poly.

Part II
Quelques notes liées au cours

Avertissement

Attention, ces notes sont un complément aux slides et à votre prise de note en CM, mais **ces notes ne constituent pas un résumé complet** des notions vues en cours. En particulier, sur certains chapitre (overfitting en particulier), elles sont incomplètes ou vides. Sur d'autres chapitres (PCA, Modèles Bayésiens) elles sont très complètes, voire vont plus loin que le CM.

11 Concepts de base, vocabulaire, modèles fondamentaux (CM1+2, séance 1)

On introduit les idées et les concepts clefs dans cette partie, en essayant de garder la technique présente mais aussi simple que possible.

11.1 Vocabulaire autour d'un modèle central: la Régression linéaire

Lors du CM on s'appuie plutôt sur un exemple classique, assez parlant, le Boston Housing dataset `https://gitlab.inria.fr/flandes/ias/-/raw/master/seance1-Vocabulaire+Regression+DescenteGradient/exemple-RegLin-CM.ipynb?inline=false`

Supposons qu'on a des paires de valeurs (x_n, y_n), par exemple x_n est le nombre d'heures passées par l'étudiant numéro n à travailler en IAS l'an dernier, et y_n est sa note à l'examen final (disons qu'au total on connaît ces valeurs pour N étudiants). On souhaite connaître la relation $f(x) = y$, ou plus exactement, pour une nouvelle observation x, on souhaite prédire le y qui correspondra. Ce genre de problème est un problème d'**apprentissage supervisé, car on connait la vraie valeur** de y_n pour un certain nombre de points de données. Plus précisément, c'est un problème de **régression car** y **prend des valeurs continues**, qui ont une **relation d'ordre** entre elles.

On peut imaginer qu'on dispose, pour chaque étudiant n, de deux données: $x_{n,1}$ le nombre d'heures passées à travailler en IAS, et $x_{n,2}$ sa note en probas l'année précédente. Dans ce cas on notera $\vec{x}_n$ pour signaler que x_n a plusieurs dimensions (on note D le nombre de dimensions). On cherchera alors la fonction $f(\vec{x}) = y$.

On égrenne ici des définition et des synonymes, illustrées sur ce cas particulier.

L'ensemble des **entrées** ou l'ensemble des **points de données** ou l'ensemble des **exemples** sont souvent notées $X = \{\vec{x}^{(1)}, \vec{x}^{(2)}, \ldots, \vec{x}^{(N)}\} = \{\vec{x}^{(n)}\}_{n\in[1,\ldots,N]}$ (ici, chaque point de donnée représente un.e étudiant.e). C'est assez naturel de noter x_{nd} la valeur stockée en ligne n, colonne d de cette matrice (tableau à deux entrées), qui est de taille (N, D). Dans ce cours, pour vous aider à visualiser les objets dont on parle, on notera parfois $A_{(N,D)}$ pour indiquer la dimension de l'objet A. Chaque point de données (*data point*) $\vec{x}_n$ pour le n-ième point de donnée (par exemple une image dans un dataset d'images), est aussi appelé **échantillon** (*sample*) ou encore **exemple** (*example*).

Les données sont en général de **dimension** notée D: un point $\vec{x}_n$ est donc un vecteur de taille D. Dans notre exemple, $D = 2$ (heures de travail, note en probas l'an dernier, ça fait 2 dimensions). Autre exemple: le nombre de pixels d'une image (si elle est en noir et blanc), ou 3 fois le nombre de pixels si l'image est en couleur (pour chaque pixel, il faut se souvenir du canal de Vert, de Rouge, de Bleu, donc ça fait 3 valeurs réelles par pixel). De façon générale, chaque dimension des données est appelée **attribut** (*feature*) ou **variable explicative** (ou encore **variable indépendante**, en modélisation statistique). Le terme *feature* est très polysémique, il est aussi utilisé pour dire autre chose, dans certains contextes. Dans notre exemple, le premier attribut est le nombre d'heures travaillées, le 2ème attribut est la note en proba l'année d'avant, etc.

Sorties: on les note y_n (ou t_n), quand il y en a. En apprentissage supervisé, les données sont sous formes de paires $(\vec{x}_n, y_n)$ (parfois on note $\vec{x}^{(n)}, y^{(n)}$, ça évite d'avoir trop d'indices). La valeur y_n contenue dans le jeu de données, qui est considérée comme fiable, est appelée **vérité terrain** (*ground truth*, GT). C'est l'**étiquette** (*label*), la **valeur à prédire**, la **cible** (*target*) (ou encore **variable dépendante**, en modélisation statistique). Cette vraie valeur est parfois aussi notée t_n (T comme True), ce qui est moins lourd que $y_n^{(GT)}$.

Le **modèle** des données: de façon générale et abstraite, on note en général $f_\theta(x)$ la fonction de sortie associée a une méthode d'Apprentissage Statistique (*Machine Learning*). Il y a souvent une différence entre **une première sortie**, notée $f_\theta(x)$, parfois même notée y, et la **prédiction** effectuée par l'algo, notée par exemple $\hat{y}$ ou $y^{predict}$. On a typiquement $\hat{y} = \sigma(f_\theta(x)) = \sigma(y)$, où σ est une **fonction de sortie** (*readout*),

parfois confondue avec une fonction d'activation. De façon générale, en maths (en probas-stats), la notation avec le chapeau se lit "valeur estimée" [sous entendu: estimée d'après des données, d'après un modèle, ...]. Donc par exemple $\hat{y}$ se lit "valeur estimée de y".

Remarque: dans l'exemple des notes et des étudiants, il est clair que pour un même nombre d'heures travaillées, il peut arriver que 2 étudiants aient des notes différentes. Le meilleur modèle imaginable ne pourra donc pas prédire correctement toutes les notes, à partir de ces données là. Il y a donc une partie "aléatoire" dans la note obtenue. Mais est-ce vraiment aléatoire? En fait, on peut se convaincre que cet aléas est plutôt un manque d'informations: si on avait beaucoup plus que 1 ou 2 données sur chaque étudiant[3], on pourrait probablement réduire de beaucoup le "hasard".

11.2 Régresssion linéaire

Le **modèle** de la régression linéaire est bien connu, au moins en dimension $D = 1$:

$$f_\theta(x) = \theta_0 + \theta_1 x \tag{10}$$

où θ_0, θ_1 sont des **paramètres réels, à ajuster**. Le premier (θ_0) s'interprète comme une ordonnée à l'origine, le second (θ_1) comme la pente de la droite $y = \theta_0 + \theta_1 x$. Dans le cas de $D > 1$, le **modèle** se généralise assez aisément:

$$f_\theta(\vec{x}) = \theta_0 + \vec{\theta_1} \cdot \vec{x} \tag{11}$$

$$= \theta_0 + \sum_d^D \theta_{1,d} x_d \tag{12}$$

Intuitivement, on a juste autant de pentes (de valeurs $\theta_{1,d}$) qu'il y a de dimensions.

Ce choix de $f_\theta(\vec{x})$ est le **modèle de la régression linéaire**. Vous voyez, vous savez déjà faire du Machine Learning !

Pour la régression, il n'y a pas de fonction de sortie (ou bien σ est la fonction identité, si vous voulez). Donc la prédiction de ce **modèle** est simplement

$$\hat{y} = f_\theta(\vec{x}) \tag{13}$$

Maintenant, il va falloir trouver un moyen astucieux de calculer des valeurs de $\theta_0, \vec{\theta_1}$ qui soient telles que "ça marche bien".

En ML, la **notation Θ – ou parfois θ – désigne l'ensemble de tous les paramètres à ajuster** dans le modèle. Pour une régression linéaire par exemple, ce sera la pente de la droite et son ordonnée à l'origine (2 paramètres si $D = 1$). En dimension D, combien a t-on de paramètres ?
Listez les paramètres, et comptez les: $\Theta = \{$ $\}$, $|\Theta| =$
Dans un réseau de neurones profond, ce sera l'ensemble de tous les poids du réseau. Le cardinal de cet ensemble, noté $|\Theta|$, peut alors être très grand.

Pour **entraîner** le modèle, on va s'appuyer sur un **jeu de données d'entrainement** (*training set*). Pour le moment, pour simplifier, on peut considérer que la totalité du tableau X participe au *train set*, c'est-à-dire qu'on utilise tout le tableau de valeurs disponibles (en fait, c'est X, T, on a aussi besoin des étiquettes, si elles existent).

Remarque: ici, un bon étudiant pourrait remarquer qu'il est inutile de faire appel à des méthodes numériques, car ce problème admet une solution exacte. Mais c'est p-e le seul cas en ML où la solution exacte peut être calculée... donc, on fait de l'optimisation approchée !

11.3 Optimisation

On va pas mal parler d'optimisation en ML, même si le ML utilise plutôt des heuristiques (des idées) d'optimisation très simples. Ici, restons intuitif. On a un modèle qui produit des prédictions. On aimerait que les prédictions $\hat{y}(x_n)$ soient proches des vraies valeurs t_n, pour les points $(\vec{x}_n, t_n)$ pour lesquels on connait la paire de valeurs.

Mathématiquement, on souhaite donc minimiser la différence entre $\hat{y}(x_n)$ et t_n. On pourrait donc chercher à minimiser $J = \sum_n |\hat{y}(x_n) - t_n|$ (valeur absolue de la différence). Mais en fait, pour des raisons techniques (on

[3] Par exemple on pourrait imaginer avoir la donnée, pour chaque étudiant.e, de son degré de connaissance de telle ou telle notion, juste avant l'examen.

en reparlera plus tard), c'est une mauvaise idée, il vaut mieux minimiser la somme *des carrés* des différences (appellée LSE, *Least Squared Error*):

$$J(\Theta, X, T) = \frac{1}{N}\sum_{n=1}^{N}(f_\theta(x_n) - t_n)^2 \tag{14}$$

On appelle ce choix la **méthode des moindres carrés** ou LSE. Il est clair que si ce truc là est petit, on est content, la différence est petite. Pourquoi mettre au carré ? Ça on verra plus tard. Précisément, on cherche à calculer:

$$\Theta^* = \text{argmin}_\Theta(J(\Theta, X, T)) \tag{15}$$

Ce qui se traduit "la meilleure valeur pour les paramètres Θ (celle qui fait que 'ça marche le mieux', notée Θ^*) est celle qui donne la plus petite valeur possible pour $J(\Theta, X, T)$":

Maintenant, si vous avez déjà fait de l'optimisation (c'est-à-dire cherché le min ou le max d'une fonction), vous savez qu'il n'y a "plus qu'à" appliquer un petit algo d'optimisation pour résoudre le problème.

11.4 Descente de Gradient: première rencontre

Voir le topo jupyter notebook: `https://gitlab.inria.fr/flandes/ias/-/raw/master/seance1-Vocabulaire+Regression+DescenteGradient/DescenteDeGradient-topo.ipynb?inline=false`

La descente de gradient est une méthode numérique d'optimisation continue qui est utile pour chercher un minimum (local). On part d'une condition initiale θ_0 arbitraire, typiquement choisie au hasard. Puis, l'itération générique qu'on doit effectuer de façon répétée est:

$$\vec{\theta} \leftarrow \vec{\theta} - \eta\vec{\nabla}_{\vec{\theta}}J(\vec{\theta}, X) \tag{16}$$

On note que le gradient joue le rôle de dérivée d'une fonction de plusieurs variables, et que c'est un vecteur de même dimension que le nombre de variables dont dépend la fonction. On note en indice ∇_θ le nom de la/les variables par rapport auxquelles on dérive.

Quelques rappels de vocabulaire:

- Modèle : $f_\theta(x)$
- Prédiction: $\hat{y} = \sigma(f_\theta(x))$
- Fit $\approx$ Ajuster les paramètres de façon optimale $\approx$ Apprendre $\approx$ optimiser une fonction coût

12 Intérêt des modèles linéaires : *feature maps* et *Kernels* (aperçu en CM2, vu à fond en CM4, séance 3)

On peut avoir besoin de **modèles plus expressifs que les modèles linéaires**. On peut voir les *feature maps* comme un *pre-processing*, tout comme la PCA et les plongements sont des sortes de *pre-processing*.

12.1 Régression polynomiale par feature map

Par exemple, on peut vouloir faire une régression polynomiale, c'est-à-dire pour un ensemble de points $(\vec{x}_n, y_n)$, trouver les coefficients du polynôme :

$$f_\theta(\vec{x}) = a_0 + \vec{a}_1\begin{pmatrix}x_1\\..\\x_D\end{pmatrix} + \vec{a}_2\begin{pmatrix}x_1^2\\..\\x_D^2\end{pmatrix} + \ldots + \vec{a}_2\begin{pmatrix}x_1^P\\..\\x_D^P\end{pmatrix} \tag{17}$$

$$= a_0 + \vec{a}_1(\vec{x})^1 + \vec{a}_2(\vec{x})^2 + \ldots + \vec{a}_P(\vec{x})^P \tag{18}$$

où P est l'ordre choisi pour le polynôme et $(\vec{x})^p$ dénote le vecteur colonne $(x_1^p, x_2^p, \ldots, x_D^p)^T$. Si on compte les paramètres, ça fait $1 + D + D \ldots + D = 1 + DP$ nombres réels.

Ce choix de modèle peut être vu comme une **expansion des attributs** (*features*) de départ: on fabrique de nouveaux attributs en combinant les précédents, on a **une fonction des features vers un nouvel espace de features**, c'est-à-dire une *feature map* (*map* signifie application en anglais). Dans l'exemple de la régression

polynomiale (ci dessus), on peut remarquer qu'on revient exactement à une régression linéaire, si on construit la *feature map* suivante (où P est un hyper-paramètre):

$$\phi(\vec{x}) = (1, x_1, \ldots, x_D, x_1^2, \ldots x_D^2, \ldots \quad \ldots x_D^P) \tag{19}$$

$$= (1, (\vec{x})^1, (\vec{x})^2, \ldots, (\vec{x})^P) \tag{20}$$

Si on compte les dimensions, ça fait $1+D+D\ldots+D = 1+DP$ nombres réels. Le modèle linéaire correspondant, pour les nouvelles données $\vec{x}' = \phi(\vec{x})$ (soit, pour le dataset complet, $X' = \phi(X)$) est:

$$f_\theta(\vec{x}') = \vec{a}'\vec{x}' \tag{21}$$

Si on compte les paramètres, il y en a autant que de dimensions, c'est-à-dire $1 + DP$ (dimensions de $\vec{a}'$). C'est bien la même chose (nos deux modèles s'entraînent exactement pareil, nous avons juste rajouter des dimensions à l'espace d'origine).

L'intérêt de cette approche est qu'on voit que **la simplicité mathématique des modèles linéaires peut être utilisée** y compris sur **des combinaisons *non* linéaires** des attributs (*features*) entre eux (x_1^2 n'est pas linéaire en x_1).

12.2 Feature map polynomiale générale

De façon plus générale, on peut construire tous les attributs issus de combinaisons d'ordre P d'un ensemble de D attributs en combinant toutes les *features* entre elles, et c'est la *feature map* polynomiale (aussi appelé Kernel polynomial). Par exemple, en $D = 3$ dimensions, pour l'ordre $P = 2$ (notre *feature map* comprends toutes les combinaisons linéaires de deux attributs), un bon choix de *feature map* polynomiale est:

$$\vec{\phi}(\vec{x}) = (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_3, \sqrt{2}x_1x_2, \sqrt{2}x_1x_3, \sqrt{2}x_2x_3, x_1^2, x_2^2, x_3^2) \tag{22}$$

Pourquoi ces coefficients $\sqrt{2}$? On peut légitimement s'en étonner. En fait, ils sont là car on obtient facilement la *feature map* ϕ en factorisant l'expression suivante (pour $P = 2$):

$$\vec{\phi}(x) \cdot \vec{\phi}(x') = (1 + \vec{x} \cdot \vec{x}')^P \tag{23}$$

Cette expression est générique pour tout P et est nommée la *feature map* standard. **Le nombre de paramètres croît rapidement avec P (explosera sûrement dès $P = 4$) et D!** Donc il faut faire attention.

12.3 Intuition générale (discutée en séance 2 == CM3)

L'idée générale est illustrée en figure 6. On passe de l'espace de départ (dimension D) à un espace de features enrichi (dimension D'). Là bas, on utilise un modèle linéaire. **Du fait de la grande dimension, le jeu de donnée est possiblement devenu *linéairement* séparable**, là où il ne l'était pas dans l'espace d'origine. C'est le pendant de la malédiction des grandes dimensions: *En dimension suffisamment élevé, tout jeu de donné devient linéairement séparable.* On pourrait parler de bénédiction des grandes dimensions, mais ... attention au sur-apprentissage (*over-fitting*) !

Si on revient ensuite dans l'espace de départ (c'est utile seulement pour les illustrations, pour l'intuition), la frontière de décision, qui était linéaire pour les $x' = \phi(x)$, ne l'est pas pour les x de dimension D.

Illustration concrète: Cette astuce permet de classifier correctement le "XOR dataset". Il suffit d'inclure la feature x_1x_2 et on pourra classifier ce dataset. En figure 7, on montre le "XOR dataset": il y a $K = 2$ classes à séparer, les points sont en $D = 2$ dimensions. Manifestement, il y a 4 cadrans, et la vérité terrain correspond à un label $y \approx XOR(x_1, x_2)$. Au centre, avec un modèle linéaire, on échoue a séparer les données. À droite, on a utilisé la feature map $\phi(\vec{x}) = (1, x_1, x_2, x_1x_2)$ (tant qu'on inclut x_1x_2 parmi les features, ça va marcher). Les données sont maintenant bien classées.

On peut tracer la frontière de décision (linéaire dans l'espace étendu) dans l'espace de départ: on matérialise la classe prédite en un point donné (x_1, x_2) par 2 couleurs (bleu clair/marron). La frontière ou séparatrice n'est pas linéaire, elle correspond à la projection de la séparatrice (linéaire) en haute dimension vers la basse dimension (un peu comme quand on regarde la section d'un cube par un plan, ça peut faire rien, ou juste un point, un triangle, ou bien un losange, et selon l'orientation du cube, ça peut faire toutes sortes de triangles/losanges plus ou moins étirés). Tout jeu de données sera linéairement séparable en au plus autant de dimensions qu'il y a de données. Il est plus facile de séparer linéairement des données dans des dimensions plus grandes, tandis que pour des dimensions plus petites, ce sera plus compliqué et on devra avoir recours à des polynômes.

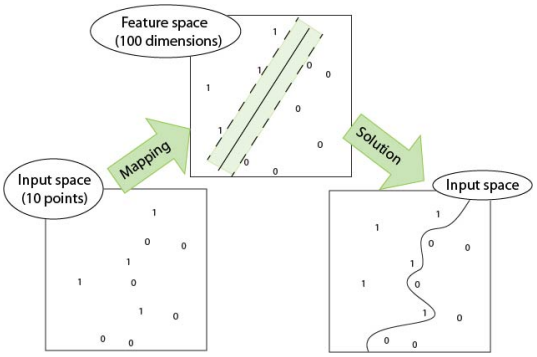


Figure 6: Illustration des *feature maps*.

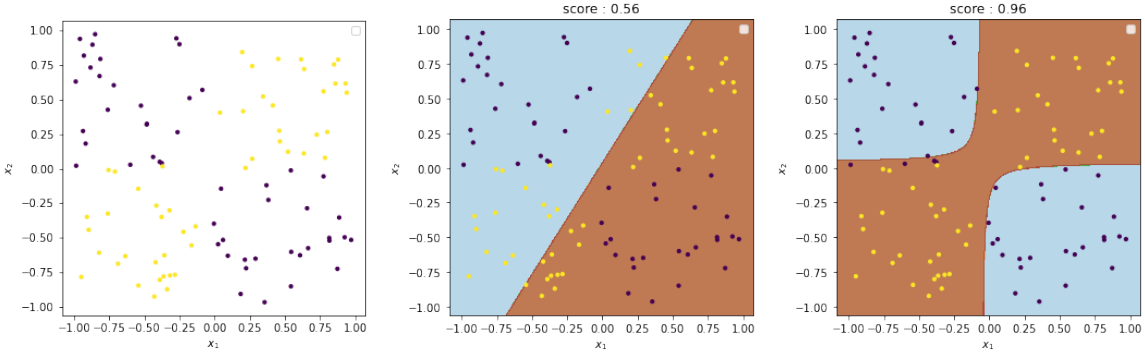


Figure 7: Illustration des *feature maps* sur le "XOR dataset".

13 Perceptron et autres classificateurs linéaires (CM3, séance 2)

13.1 Motivation

On a vu la régression, c.a.d. le cas où l'étiquette (*label*) à prédire est une valeur continue. Si l'étiquette est **discrète** (représente des **catégories**, des **classes**), on parle de **classification**.

Par exemple, si on doit identifier des bars et des saumons dans une pêcherie. Ou des chiens et des chats sur des photos. etc. Si il y a $K = 2$ classes, on parle de **classification binaire**. Si $K \geq 3$, on parle de classification **multi-classe**.

13.2 Classifieurs linéaires

Les classificateurs **linéaires** sont les analogues en classification de la régression **linéaire**: on modélise la séparation entre classes par une frontière **linéaire** (donc un **hyper-plan** de dimension $d-1$, pour un espace de dimension d).

Par exemple pour $d = 2$ attributs, c'est une droite **séparatrice**.

Les exemples situés de part et d'autre du plan ont comme étiquette prédite une classe ou l'autre (on traite le cas $K = 2$ pour le moment).

Notations de ce cours ci: les vecteurs avec prime w', x' sont les vecteurs naturels de l'espace à D dimensions, et les vecteurs augmentés (avec le 1 devant pour les données x' et avec le paramètre w_0 en haut pour les paramètres w') sont notés sans prime.

$$\vec{w} = \begin{pmatrix} w_0 \\ \begin{pmatrix} \vec{w}' \end{pmatrix} \end{pmatrix} = \begin{pmatrix} w_0 \\ w_1 \\ \dots \\ w_D \end{pmatrix} \quad ; \quad \vec{x} = \begin{pmatrix} 1 \\ \begin{pmatrix} \vec{x}' \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 1 \\ x_1 \\ \dots \\ x_D \end{pmatrix} \tag{24}$$

Remarquez que seul les vecteurs w', x' en $D = 2$ peuvent être représentés directement dans le plan comme des vecteurs "normaux". La coordonnée w_0 de $\vec{w}$ contraint la position de la droite $\mathcal{D}(\vec{w})$ dans le plan, alors qu'un vecteur est en général libre d'être placé n'importe où (son point de départ est libre).

On voit en TDs 3.1 et 3.5 les formules qui donnent la distance (signée) entre un point et un plan, dans le cas général. On s'appuie sur ces formules pour établir la classe prédite par un algo de classification linéaire.

Concrètement, on s'arrange en général pour que le vecteur directeur du plan soit normé, c.a.d. que $||\vec{w}'||_2 = 1$, et dans ce cas, la distance entre un point $\vec{x}'$ [4] et le plan [5] est donnée par $\vec{w}' \cdot \vec{x}' + w_0$, qui se ré-écrit $\vec{w} \cdot \vec{x}$.

Un modèle linéaire s'écrit donc $f_w(\vec{x}) = \vec{w} \cdot \vec{x}$.

La prédiction associée est $\hat{y}_n = signe(\vec{w} \cdot \vec{x}_n)$, en supposant que les labels sont encodés sous la forme $t_n \in \{-1, +1\}$.

Il faut maintenant trouver un algorithme qui permette de trouver des paramètres $\vec{w}$ optimaux en un certain sens (à définir).

13.3 Algorithme du Percpetron

Intuition: on a détaillé l'intuition en CM. En résumé, on se convainc d'abord qu'il faut minimiser le nombre de points mal classés, et même faire en sorte que les points mal classés le soient "le moins possible". En conclusion, il faut minimiser **la somme des distances entre pointa mal classés et plan**. Ceci se traduit par la fonction coût suivante[6]

$$J(\Theta, X, T) = \frac{1}{N} \sum_{n}^{N} \max(0, -(\vec{w} \cdot \vec{x}_n t_n) \tag{25}$$

On peut vérifier que cette formule calcule la somme des distances au plan des points mal classés. En effet, on a:

$$\max(0, -(\vec{w} \cdot \vec{x}_n t_n) \begin{cases} 0 & \text{si bien classé} \\ > 0 & \text{Sinon} \end{cases} \tag{26}$$

On calcule que $\nabla_\Theta J(\Theta, X) = -\frac{1}{N} \sum_{n \in \text{mal classes}} \vec{x}_n t_n$.

On rappelle la formule de descente de gradient: $\vec{\Theta} \leftarrow \vec{\Theta} - \eta \vec{\nabla}_{\vec{\Theta}} J(\Theta, X, T)$

Les mises à jour des poids sont donc

$$\vec{w} \leftarrow \vec{w} + \eta \frac{1}{N} \sum_{n \in \text{mal classes}} \vec{x}_n t_n \tag{27}$$

on remarque dans l'exemple en jupyter-notebook que les corrections appliquées à la séparatrice vont dans le bon sens.

13.4 3 stratégies d'optimisation (SGD, mini batches, full Batch)

Il y a **plusieurs stratégies** pour minimiser une fonction coût du style de celle de la régression linéaire ou du perceptron. En notant $\mathcal{L}(\Theta, X, T)_n$ le **terme de perte lié à 1 seul exemple** (par exemple $\mathcal{L}(\Theta, X, T)_n = (\hat{y}_n - t_n)^2 = (\vec{w}\vec{x}_n - t_n)^2$ pour la reg. lin.), on peut écrire la fonction coût standard comme la moyenne sur tous les exemples:

$$J_N(\Theta, X, T) = \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}(\Theta, X, T)_n \tag{28}$$

En minimisant J_N, on adopte la stratégie de base, standard, dite aussi "*full Batch*" Les mises à jour par descente de gradient s'écriront:

$$\vec{\theta} \leftarrow \vec{\theta} - \eta \nabla J_N \tag{29}$$

Chaque mise à jour représente un **passage en revue de chacun des exemples** d'entraînement, c.a.d. une **époque**.

Descente de Gradient Stochastique (*Stochastic Gradient Descent*): On peut choisir de minimiser plutôt le coût pour 1 exemple choisi au hasard. On défini

$$J_{n,sgd}(\Theta, X, T) = \frac{1}{1} \mathcal{L}(\Theta, X, T)_n \tag{30}$$

[4] un point $\vec{x}'$ ou son vecteur augmenté avec un en attribut supplémentaire, $\vec{x} = (1, \vec{x}'$

[5] plan repéré par $w = (w_0, \vec{w}')$

[6] **NOTE: on ajoute le $\frac{1}{N}$, qui est une bonne habitude.. on l'avait oublié dans les slides..**

on tire au hasard l'indice $n \in [1, .., N]$, puis on effectue la mise à jour des paramètres ainsi:

$$\vec{\theta} \leftarrow \vec{\theta} - \eta \nabla J_{n,sgd} \tag{31}$$

Il faudra tirer au hasard (et mettre à jour) N fois pour avoir vu N exemples (et parfois plusieurs fois le même, peu importe) pour avoir complété une époque.

Stratégie intermédiaire, dites des petits paquets (*mini-batches*): On définit le coût sur m exemples tirés au hasard (on permute les indices des exemples, par exemple):

$$J_{m,mb}(\Theta, X, T) = \frac{1}{m} \sum_{n=1...m} \mathcal{L}(\Theta, X, T)_n \tag{32}$$

on effectuera la mise à jour des paramètres ainsi:

$$\vec{\theta} \leftarrow \vec{\theta} - \eta \nabla J_{m,mb} \tag{33}$$

Il faudra tirer au hasard (et mettre à jour) $N/m \in \mathbb{N}$ fois pour avoir vu N exemples différents pour avoir complété une époque. À chaque époque, on permute les exemples, de sorte que les mini-bacthes n'arrivent pas dans le même ordre et la même composition d'exemples à chaque époque. On peut aussi piocher m exemples au hasard à chaque mini-batch.. ce n'est pas crucial

Version Online: c'est comme la SGD mais en prenant les exemples toujours dans le même ordre. C'est moins bien car sujet à cycles récurrents, ce qui n'est pas le cas d'une version aléatoire.

13.4.1 Intérêt des stratégies

C'est discuté en CM. En très bref, l'aléas introduit par les méthodes stochastiques évite de rester piéger dans un minimum local de J_N, ce qui est en général mauvais pour la généralisation.

13.5 Classifications multi-classe

Si $K > 2$, on parle de Classification multi-classe. Nous représenterons alors la classe par un vecteur one-hot:

$$t_n = \vec{e}_k = (0, ..., 0, \underbrace{1}_{\text{k-ième classe}}, 0, ..., 0)$$

L'idée est ensuite d'avoir K plans, chaque plan $\vec{w}_k$ servant à déterminer si un point $\vec{x}_n$ est de la classe k ou bien pas de la classe k. C'est la stratégie **One-versus-rest**.

Un point est affecté à la classe pour laquelle il est situé le plus possible à l'intérieur, c.a.d quand la distance *signée* point-plan est la plus grande (soit que ce soit positif, et alors il faut que le plan soit le plus loin possible dans le domaine définit par le plan, soit que ce soit négatif pour toutes les classes, et alors il faut qu'il soit "le moins négatif" possible). La classe gagnante est donc $k^* = argmax_k(\vec{w}_k \vec{x}_n)$, et la prédiction sous forme de vecteur one-hot est donc $\hat{y}_n = \vec{e}_{k^*}$.

Au lieu de D paramètres, on a donc DK paramètres, D pour chacun des K plans.

Pour l'apprentissage, on remplace $signe(f(x))$ par $softmax(f(x))$:

$$y_k^{(n)} = \frac{\exp(-\vec{w}_k.\vec{x}^{(n)})}{\sum_{k'} \exp(-\vec{w}_{k'}.\vec{x}^{(n)})}$$

Autre stratégie: **One-versus-one** – il y a alors $(K(K-1)/2)$ plans !

13.6 Support Vector Machine (SVM)

Idée: On définit une marge et on exige sa maximisation. Seuls quelques points vont finalement "porter" l'hyper-plan, ce sont les vecteurs supports. Les points mal classés sont tous supports (dans la marge), on paye un prix pour ceux là. En 2D, dans un cas séparable linéairement, il n'y que trois vecteurs suports, qui définissent le plan séparateur.

14 Overfitting, généralisation (CM4-5, séance 3-4)

Ce chapitre est très important mais les slides sont suffisantes. Les notions clefs sont:

Overfitting
Cross-Validation
Séparation en 3 ensembles: **train, validation, test**.
Complexité du modèle
Régularisation
Métriques de performance
Équilibrage des classes

15 Pre-traitements et encodages (CM6, séance 5)

15.1 Standardisation

De façon Sous jacente (implicite), on a besoin de choisir une distance entre les points de données. La standardisation est une façon de déformer l'espace pour que chaque direction (attribut, feature) compte autant.

Intuition: si dans les données par exemple on a le revenu mensuel, le patrimoine en euros, le poids en kg et la taille humaine exprimée en mètres, si on ne standardise pas, alors le patrimoine, qui peut aller des milliers aux millions (ou plus), comptera beaucoup plus dans la mesure de la distance entre deux points de données (deux personnes) que la taille en mètres, qui variera entre 1 et 2. Si on ne standardise pas ce genre de données, ça revient à essentiellement effacer la feature "taille en mètres". Inversement si la fortune (patrimoine) est exprimée en milliers de milliards d'euros, alors ce sera 0 pour quasi tout le monde, les variations seront de l'ordre de 0.00...01 et cette *feature* n'aura quasi aucun impact sur la distance entre deux points. D'où l'**importance des pré-processings**.

Encore une explication intuitive: Sur le cas poids/taille (vu au dessus), si la taille est en millimètres les distances ne vont pas être les mêmes qu'avec des centimètre ni les même qu'avec des mètres. Donc l'algo sera biaisé car la feature de taille sera favorisée (bcp plus grande) par rapport au poids.

Autre astuce:
Si on veut qu'une donnée compte plus, on peut par exemple multiplier cette donnée (x2 par exemple) OU faire une copie de la variable pour qu'elle apparaisse 2 fois. L'écart-type de cette feature sera doublé, elle "pèsera" alors plus dans les calculs (pour la plupart des algos en tout cas).

Si les données ne sont **pas homogènes**, le choix le plus courant est de **standardiser**, c.a.d de **soustraire la moyenne et diviser par l'écart-type**, le long de chaque dimension.

$$\forall d \in [1, ..., D]: \quad x_{n,d} \to \frac{x_{n,d} - \langle x_d \rangle}{\sigma_d} \tag{34}$$

$$\text{avec} \qquad \langle x_d \rangle = \sum_{n=1}^{N} x_{n,d} \qquad \text{et} \qquad \sigma_d = \langle (x_{n,d} - \langle x_d \rangle)^2 \rangle^{1/2} = \left(\sum_{n=1}^{N} (x_{n,d} - \langle x_d \rangle)^2 \right)^{1/2} \tag{35}$$

Où σ_d est l'écart type empirique de la feature numéro d (c.a.d. l'écart type empirique de l'ensemble des réels $x_{nd}, n = 1 \dots N$). En sortie, on obtient automatiquement des données dont la moyenne est 0 (le long de chaque direction), et l'écart-type est de 1 (le long de chaque direction). On appelle cela une variable "**centrée et réduite**" (centrée=moyenne à 0, réduite=écart type à 1). D'une certaine façon, cela fait que chaque feature (chaque dimension) "pèse" autant que n'importe quelle autre.

Opération de standardisation sous forme vectorielle/algébrique :

$$X_{N,D} \to X'_{N,D} = \frac{X_{N,D} - \langle X \rangle}{\sigma_X}$$

C'est une écriture informelle, où $\langle X \rangle$ et σ_X sont de dimension D. X' vérifie : $\langle X'_d \rangle = 0, \sigma_{X'_d} = 1$

Standardiser n'est pas la meilleure solution si la distribution de la feature d est très différente de 1 ou 2 paquets gaussiens proches. En particulier si les données sont concentrées autour des valeurs minimale et maximale. Dans ces cas là, il est préférable que **le facteur d'échelle appliqué soit le min-max**: on va diviser les données par $\max_n(x_{nd}) - \min_n(x_{nd})$. Il n'y a alors **pas lieu de centrer les données** par rapport à leur moyenne. En choisissant d'envoyer les valeurs de l'attribut numéro d vers l'intervalle [0,1] (ou [-1,1]), on aura contraint ses valeurs à couvrir la plage de valeurs choisie. Si on souhaite atteindre $[0, 1]$:

$$x_{n,d} \to \frac{x_{n,d} - \min_n(x_{nd})}{\max_n(x_{nd}) - \min_n(x_{nd})} \tag{36}$$

15.2 Vecteurs one-hot

Rappel: Le nombre a_{nk} dénote le fait que le point n soit affecté au cluster k (ou pas affecté à celui là). On a
$a_{nk} = 1$ lorsque le point $n \in$ cluster k, et
$a_{nk} = 0$ lorsque le point $n \notin$ cluster k.

Le vecteur $\vec{a}_n$ est un vecteur one-hot, on peut le noter $(0, .., 1, ..., 0)$ où le 1 est à la k-ieme place. cf. `https://fr.wikipedia.org/wiki/Encodage_one-hot`.

Remarque: si on note k^{GT} le label associé à l'exemple n, on peut noter avec un symbole delta de Kronecker:

$$a_{nk} = \delta(k, k^{GT}) = \begin{cases} 1 \text{ si } k = k^{GT} \\ 0 \text{ si } k \neq k^{GT} \end{cases} \tag{37}$$

Cet encodage est très approprié lorsqu'une dimension (feature) dénote le choix d'une catégorie parmi de plusieurs catégories, et que celles ci n'ont aucune proximité ou relation particulière. Par exemple si il y a 10 catégories d'images (avion, voiture, chien, chat, ...), il n'y a aucune raison de noter 1 pour la première, 2 pour la seconde, etc: en effet si on faisait ça, ça induirait une sorte de proximité plus grande entre les catégories 1 et 2 qu'entre les catégories 1 et 10, alors que ça n'a aucun sens.

Dans ce genre de cas, l'encodage one-hot est tout indiqué !

16 Réduction dimensionnelle : cas de l'Analyse en Composantes Principales, ACP (Principal Components Analysis, PCA) (aperçu en CM6, vu à fond plus tard)

L'analyse en composantes principales (en anglais, *Principal Components Analysis*, PCA) peut-être vue comme un des nombreux pré-traitements disponibles dans la panoplie du Machine-Learner. Mais c'est un outil en fait beaucoup plus général et puissant, qui est plus ancien que l'avènement des méthodes de ML modernes.

En dépit de son âge, cette méthode n'a pas tellement vieilli: de par son élégance, ou son côté "bien fondée formellement", elle reste d'actualité.

La PCA est une méthode d'**apprentissage non supervisé**. Elle fait partie des méthodes dites de **réduction dimensionnelle** (*dimensional reduction*).

La PCA peut aussi être vue comme une méthode de visualisation de données en grande dimension. Cependant, ce serait ignorer sa puissance de la limiter à ce rôle. D'autres méthodes, moins contrôlées mais assez puissantes, comme *t-SNE*, sont bien adaptées pour la visualisation de données en grande D.

La PCA peut aussi être vue comme une méthode de compression des données. La reconstruction (décompression) est très facile.

16.1 Intuition sur : réduction dimensionnelle

De façon générale (et non seulement pour la PCA), les méthodes de réduction dimensionnelle consistent à réduire la dimension de l'espace d'entrée (l'espace des features, de dimension D), vers une autre dimension, plus petite, D' (ou p). Concrètement, cela consiste à chercher une application non injective Φ:

$$\Phi : \mathbb{R}^D \longrightarrow \mathbb{R}^{D'}, \text{ avec} D' < D \tag{38}$$

Pourquoi est-ce intéressant? Il y a différentes raisons, qui se recoupent entre elles. Réduire la dimension des donnée:

- Permet de supprimer les redondances dans l'information d'entrée. Par exemple, lorsque 2 attributs sont identiques en valeur, ou de valeur proportionnelle, il n'y a pas d'intérêt à les garder en doublon (c'est même nuisible, a priori).
- Permet de réduire le "bruit" (c'est le nom qu'on donne aux informations "inutiles"... c'est une vaste question de savoir ce que ça veut dire, précisément, et il est très difficile de distinguer le bruit du signal).
- On évite ainsi la malédiction des grandes dimensions. En effet, en général (mais pas nécessairement), plus les dimensions sont grandes, plus on a de paramètres à ajuster et on se retrouve facilement dans des cas de sur-apprentissage.
- On évite donc le sur-apprentissage: en effet, qui dit moins de dimensions en entrée dit, en général, moins de paramètres dans le modèle.
- Cela aide donc l'apprentissage: en termes de vitesse (a priori, c'est le cas, on peut probablement imaginer des cas pathologiques où ça ralentit l'apprentissage).
- Cela aide donc l'apprentissage: en termes de performance: ça dépend. On peut limiter la performance en supprimant les données, mais on peut aussi l'améliorer par la diminution du sur-apprentissage.

16.2 PCA: idée

Ici on ne verra qu'une réduction dimensionnelle: la PCA.

Dans ce cas particulier, l'application Φ est une projection (application linéaire) de l'espace $\mathbb{R}^D$ dans le sous-espace $\mathbb{R}^{D'}$. On projette les données sur un hyper-plan de dimension D', qui a une certaine orientation dans l'espace parent $\mathbb{R}^D$. Dans la figure 8, on voit ce que cela peut signifier, dans un cas à petite dimension, ou on peut encore dessiner. On voit aussi, ou bien on rappelle que, une projection est aussi le choix d'un nombre D' de combinaisons linéaires des D attributs d'entrée. Chaque dimension/nouvel attribut, après PCA, sera une combinaison linéaire des attributs d'entrée.

On notera en général $\vec{x}' = P\vec{x}$ les données après projection ($P_{(D,D')}$ étant la matrice de projection, ou la matrice des coefficients des combinaisons linéaires, si vous voulez). Dans le cas particulier où $D' = 1$, x' est un nombre et $x' = \vec{x}.\vec{P}$ où . représente le produit scalaire.

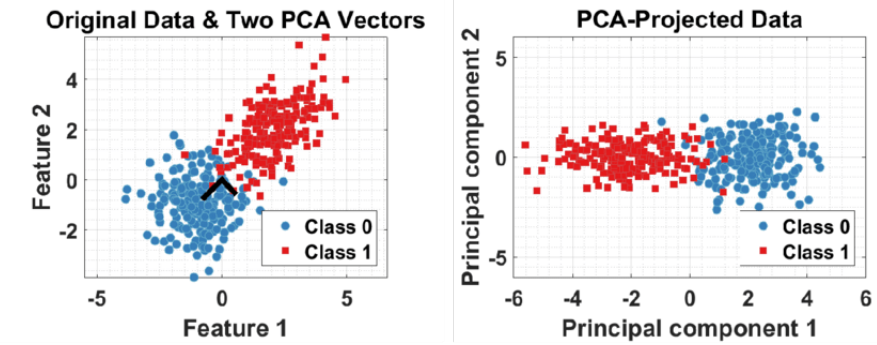


Figure 8: PCA de données à $D \geq 2$ dimensions (on n'en voit que 2, les 2 premières, mais on peut imaginer qu'il y en a d'autres, orthogonales aux premières) vers $D' = 2$ dimensions. Si on imagine le cas $D = 2$, on fait alors juste une rotation. Si on avait choisi $D' = 1$, il ne resterait pour représenter les données que l'axe des abscisses, celui de la première composante: dans l'image de droite, il faudrait imaginer que l'axe des ordonnées a été écrasé, qu'il n'en reste que la valeur moyenne (ou 0).

La question qui se pose maintenant est: comment choisir un bon hyper-plan sur lequel ou voudrait projeter ? Le choix fait, dans le cadre de la PCA, est de prendre l'hyperplan **tel que la variance des données, après projection, est maximale**. On favorise donc les dimensions de grande variance (et leurs combinaisons linéaires). On verra que **ce choix est équivalent à un autre choix**, qui serait celui de vouloir **minimiser l'erreur de reconstruction** (en norme L2, c'est-à-dire de minimiser l'erreur quadratique moyenne de reconstruction).

L'idée (naïve) ici est donc de **garder les attributs qui varient le plus** (car se sont ceux qui portent le plus d'information). Par ailleurs, si les données sont très hétérogènes, on peut les standardiser un minimum avant d'effectuer la PCA (pas sûr pendant le cours).

16.3 PCA: démonstration du bien-fondé de la méthode

16.3.1 Variance des données après projection

Calculons donc la variance des données, après projection. Pour commencer, on ne s'intéresse qu'à une direction de projection. Soit $\vec{u}_i$ le vecteur qui caractérise l'axe de la projection. Les données projetées deviennent donc de simples nombres, qu'on peut noter $x'_n = \vec{u}_i^T \vec{x}_n = \sum_d u_{id} x_{nd}$.

On va prouver que la variance des données le long de cet axe, $\sigma_i'^2$, est donnée par: $\sigma_i'^2 = \vec{u}_i^T C \vec{u}_i$, où C est la matrice de covariance des données. (L'idée de la démonstration est de se concentrer sur une direction, ici $\vec{u}_i$). On a déjà parlé de la covariance, mais on est sympa, on rappelle la formule:

$$C = \frac{1}{N} \sum_n^N (\vec{x}_n - \langle \vec{x} \rangle)(\vec{x}_n - \langle \vec{x} \rangle)^T \tag{39}$$

Pour rappel, C est clairement une matrice de taille (D, D) (somme de N produits matriciels de matrices $(D, 1)$ par matrices $(1, D)$) et $\langle \vec{x} \rangle$ représente le vecteur moyenen de $\vec{x}$ (la moyenne empirique). On calcule la variance

des données après projection sur $\vec{u}_i$, notée $\sigma_i'^2$:

$$\sigma_i'^2 = \frac{1}{N}\sum_n (x'_n - \langle x' \rangle)^2 \qquad = \text{Définition de la variance} \tag{40}$$

$$= \frac{1}{N}\sum_n (\vec{u}_i^T \vec{x}_n - \langle \vec{u}_i^T \vec{x} \rangle)^2 \tag{41}$$

$$= \frac{1}{N}\sum_n (\vec{u}_i^T \vec{x}_n - \langle \vec{u}_i^T \vec{x} \rangle)(\vec{u}_i^T \vec{x}_n - \langle \vec{u}_i^T \vec{x} \rangle)^T \tag{42}$$

Où on a utilisé que $a_i^2 = a_i a_i^T$, même si a_i est de taille (1,1).

$$= \frac{1}{N}\sum_n (\vec{u}_i^T \vec{x}_n - \langle \vec{u}_i^T \vec{x} \rangle)(\vec{x}_n - \langle \vec{x} \rangle)^T (\vec{u}_i^T)^T \tag{43}$$

On a utilisé que $(AB)^T = B^T A^T$.

$$= \frac{1}{N}\sum_n \vec{u}_i^T (\vec{x}_n - \langle \vec{x} \rangle)(\vec{x}_n - \langle \vec{x} \rangle)^T \vec{u}_i \tag{44}$$

$$= \vec{u}_i^T \quad \frac{1}{N}\sum_n (\vec{x}_n - \langle \vec{x} \rangle)(\vec{x}_n - \langle \vec{x} \rangle)^T \quad \vec{u}_i \tag{45}$$

$$\sigma_i'^2 = \vec{u}_i^T \quad C \quad \vec{u}_i \tag{46}$$

CQFD.

16.3.2 Calcul de la meilleure direction

Comme on l'a dit, **le choix fait dans la méthode de la PCA est celui de maximiser la variance des données projetées, donc trouver le $\vec{u}_i$ qui maximise $\sigma_i'^2$**. A priori, on pourrait écrire que le meilleur vecteur de projection est donc :

$$(?) \quad \vec{u}_i^* = \underset{\vec{u}_i \in \mathbb{R}^D}{\text{argmax}} \left(u_i^T C \vec{u}_i \right) \quad (?) \tag{47}$$

Cependant, si on résout ce problème d'optimisation, on obtient la solution triviale et non intéressante, qu'il faut prendre un vecteur $\vec{u}_i$ de la norme la plus grande possible (une matrice de Covariance est par définition semi-définie positive, c.a.d. que ses valeurs propres sont positives ou nulles). Ce n'est pas intéressant, car on fait une projection, ce qui nous intéresse c'est de trouver le bon angle de projection, pas de multiplier nos données par ∞ pour augmenter leur variance.

On décide donc de se restreindre aux vecteurs de norme 1, c'est-à-dire de ne regarder que l'orientation de $\vec{u}_i$: on impose $\vec{u}_i^2 = 1$ On traite donc du problème d'optimisation sous contrainte suivant:

$$\vec{u}_i^* = \underset{\vec{u}_i \in \mathbb{R}^D, \vec{u}_i^2 = 1}{\text{argmax}} \left(u_i^T C \vec{u}_i \right) \tag{48}$$

Qui peut se réécrire, en attribuant à la contrainte $u_i^2 = 1 \Leftrightarrow (1 - u_i^2) = 0$ le multiplicateur[7] λ:

$$\vec{u}_i^* = \underset{\vec{u}_i \in \mathbb{R}^D}{\text{argmax}} \left(u_i^T C \vec{u}_i + \lambda(1 - \vec{u}_i^2) \right) \tag{49}$$

On note:

$$F(\vec{u}_i) = u_i^T C \vec{u}_i + \lambda(1 - \vec{u}_i^2) \tag{50}$$

$$\vec{u}_i^* = \underset{\vec{u}_i \in \mathbb{R}^D}{\text{argmax}} \left(F(\vec{u}_i) \right) \tag{51}$$

Ce problème de maximisation est assez simple, et il se trouve qu'on peut le résoudre exactement. La méthode est classique: on cherche le zéro de la fonction que est dans le argmax, c'est-à-dire on cherche à résoudre $\vec{\nabla}_{\vec{u}_i} F(\vec{u}_i) = \vec{0}$. Ce n'est pas évident, mais le gradient de $u^T C u$ se calcule, pour une matrice C symmétrique (et donc telle que $C^T = C$): $\vec{\nabla}_{\vec{u}}(\vec{u}^T C \vec{u}) = 2C\vec{u}$. En fait, pour faire ce calcul, on repart de la définition intiale,

[7] Ici, le signe à mettre devant la contrainte s'obtient en réfléchissant sur la direction qu'on veut contenir: en fait, ce qu'on veut, c'est que $(1 - u_i^2) \geq 0$, mais cette inégalité va être saturée car les plus grand u_i^2 sont favorisés. D'où l'ajout de $\lambda(1 - \vec{u}_i^2)$ à la fonction, et non pas son opposé, $\lambda(\vec{u}_i^2 - 1)$.

c'est-à-dire de $\vec{u}^T C\vec{u} = \frac{1}{N}\sum_n^N (\vec{u}^T\vec{x} - \langle\vec{u}^T\vec{x}\rangle)^2$. Pour alléger la notation, on va supposer, temporairement, que les données sont centrées, c'est-à-dire que $\langle\vec{x}\rangle = \vec{0}$. Ça ne change pas la valeur de la matrice de covariance, et ça allège l'écriture.

$$\vec{\nabla}_{\vec{u}}(\vec{u}^T C\vec{u}) = \vec{\nabla}_{\vec{u}}\frac{1}{N}\sum_n^N(\vec{u}^T\vec{x}_n - \cancel{\langle\vec{u}^T\vec{x}_n\rangle})^2 \tag{52}$$

$$= \frac{1}{N}\sum_n^N \vec{\nabla}_{\vec{u}}(\vec{u}^T\vec{x}_n)^2 \tag{53}$$

$$= \frac{1}{N}\sum_n^N 2\vec{x}(\vec{u}^T\vec{x}_n)^1 \qquad \text{car } \frac{\partial}{\partial s}f^2(s) = 2\frac{\partial f}{\partial s}f(s) \tag{54}$$

$$\text{Puisque } \vec{u}^T\vec{x} \text{ est un nombre,} \vec{u}^T\vec{x}_n = (\vec{u}^T\vec{x})^T = \vec{x}_n^T\vec{u} \tag{55}$$

$$= \frac{1}{N}\sum_n^N 2\vec{x}_n(\vec{x}_n^T\vec{u}) \tag{56}$$

$$= 2\frac{1}{N}\sum_n^N(\vec{x}_n\vec{x}_n^T)\vec{u} \tag{57}$$

$$= 2C\vec{u} \tag{58}$$

Par ailleurs, il est facile de vérifier que $\vec{\nabla}_{\vec{u}}\lambda\vec{u}^2 = 2\lambda\vec{u}$. On peut donc terminer la recherche du argmax($F(\vec{u}_i)$):

$$\vec{\nabla}_{\vec{u}_i}F(\vec{u}_i) = \vec{0} \Leftrightarrow \tag{59}$$

$$\vec{0} = 2C\vec{u} - 2\lambda\vec{u} \tag{60}$$

$$C\vec{u} = \lambda\vec{u} \tag{61}$$

C'est l'équation aux valeurs propres ! Comme c'est joli ! C'est un résultat assez profond. Il y a donc **autant de maxima locaux de la fonction $F(\vec{u}_i)$ qu'il y a de couples** $(\lambda_i, \vec{u}_i$ (couples de valeur propre-vecteur propre). La norme des vecteurs propre est imposée par notre contrainte $\vec{u}^2 = 1$. La direction (le signe devant $\vec{u}$) est arbitraire, ce qui est normal pour une projection, car une projection dépend de la droite sur laquelle on projète, mais pas de son orientation. Par contre, il y a plein de valeurs propres... laquelle faut il prendre ? Hé bien, il faut prendre celle qui réalise le maximum global de $F(\vec{u}_i)$.

Maintenant qu'on sait qu'on s'intéresse aux vecteurs $\vec{u}$ de norme 1 et tels que $C\vec{u} = \lambda\vec{u}$ (c'est-à-dire les vecteurs propres), on peut réécrire F, en prenant un couple $(\lambda_i, \vec{u}_i)$ arbitraire:

$$F(\vec{u}_i)_{|\vec{\nabla}_{\vec{u}_i}F(\vec{u}_i)=\vec{0}} = \vec{u}_i^T C\vec{u}_i + \lambda(1 - \vec{u}_i^2) \tag{62}$$

$$= \vec{u}_i^T\lambda_i\vec{u}_i + 0 \tag{63}$$

$$= \lambda_i\vec{u}_i^T\vec{u}_i \tag{64}$$

$$= \lambda_i \tag{65}$$

Plutôt propre, non ? Donc en fait, **la variance des données projetées est exactement égale à la valeur propre correspondant à la direction propre qu'on choisit pour projeter**. Une autre façon de le voir est de juste calculer la variance projetée, sans passer par F: $\sigma_i^2 = \vec{u_i}^T(C\vec{u_i}) = \vec{u_i}^T\lambda_i\vec{u_i} = \lambda_i\vec{u_i}^2 = \lambda_i$.

Conclusion: la meilleure direction, parmi toutes les directions, est celle donnée par le vecteur propre qui est associé à la plus grande valeur propre.

16.3.3 Directions suivantes

En pratique, on ne souhaite pas conserver que 1 composante, mais plutôt projeter depuis un espace de dimension D vers un espace de dimension $D', 1 < D' < D$. On projette donc non pas sur une droite (un seul vecteur) mais plutôt sur un hyper-plan, dont l'orientation est définie par les vecteurs qui sont dans ce plan. Pensez par exemple à une paire de vecteurs (non colinéaires) dans un espace à $D = 3$: ces deux vecteurs définissent naturellement un plan (à $D' = 2$ dimensions).

Les données **après projection sur un hyper-plan** de dimension $D'2$, par exemple, s'écrivent de la façon suivante. Appelons $\vec{u}_1 \in \mathbb{R}^D, \vec{u}_2 \in \mathbb{R}^D$ les deux vecteurs qui définissent ce plan. Appelons $\vec{e}_1 = (1, 0), \vec{e}_2 = (0, 1)$

les deux vecteurs de la base interne au plan (ces vecteurs définissent un repère interne au plan, qui permet de se promener dedans). Alors la projection s'écrit:

$$\vec{x}' = (\vec{x}^T \vec{u}_1)\vec{e}_1 + (\vec{x}^T \vec{u}_2)\vec{e}_2 \tag{66}$$

L'écriture générale est $\vec{x}' = \sum_{d'=1}^{D'} (\vec{x} \cdot \vec{u}_{d'})\vec{e}_{d'}$.

Quels sont les directions qui définissent cet hyper-plan ? On ne refait pas tout le raisonnement, mais au vu de la partie précédente, **la 2ème meilleure direction est celle donnée par le vecteur propre qui est associé à la plus grande valeur propre**. Et ainsi de suite pour la 3ème, la 4ème, etc.

On aboutit alors à la remarque suivante. Si on diagonalise la matrice de covariance, C, on a : $C = U\Lambda U^{-1}$, avec Λ (se lit Lambda majuscule, comme λ) la matrice diagonale des valeurs propres, et U la matrice de passage vers l'espace où C est diagonalisée.

$$\Lambda = \begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_D \end{pmatrix} \qquad U = \left(\begin{pmatrix} \cdot \\ \vec{u}_1 \\ \cdot \end{pmatrix} \quad \begin{pmatrix} \cdot \\ \vec{u}_2 \\ \cdot \end{pmatrix} \quad ... \quad \begin{pmatrix} \cdot \\ \vec{u}_D \\ \cdot \end{pmatrix} \right) \tag{67}$$

On ne rentre pas dans les détails, mais il est légitime de supposer qu'une matrice de covariance de données "normales" sera diagonalisable [8] , avec tous les vecteurs propres distincts deux à deux. Par définition de C, elle est semi-définie positive, c'est-à-dire que toutes ses valeurs propres sont positives ou nulles. Comme toujours lors d'une diagonalisation, U **est constituée des vecteurs propres mis côte à côte**, dans le même ordre que les λ_i dans Λ. Ici, on choisit d'**ordonner les λ_i par valeur décroissante** (elles sont toutes positives et réelles).

La matrice de projection sur l'hyper-plan de dimension D' s'obtient alors (souvenez vous qu'on a trié les λ_i par ordre décroissant) en gardant les D' premières directions propres:

$$P = \left(\begin{pmatrix} \cdot \\ \vec{u}_1 \\ \cdot \end{pmatrix} \quad \begin{pmatrix} \cdot \\ \vec{u}_2 \\ \cdot \end{pmatrix} \quad ... \quad \begin{pmatrix} \cdot \\ \vec{u}_{D'} \\ \cdot \end{pmatrix} \right) \tag{68}$$

La matrice P n'est pas carrée, elle est de taille (D, D') (D lignes, D' colonnes). Comme on l'a dit plus haut, la donnée $\vec{x}_n$ projetée s'obtient alors ainsi: $\vec{x}'_n = \sum_{d'=1}^{D'} (\vec{u}_{d'} \cdot \vec{x}_n)\vec{e}_{d'}$. Ceci est en fait l'écriture d'une multiplication matricielle:

$$\vec{x}_{n,transformed} = (\vec{x}_n)' = (\vec{x}_n - \langle \vec{x} \rangle) \cdot P \tag{69}$$

Ces vecteurs sont de dimension D'.

16.3.4 Décompression

La transformée inverse s'obtient ainsi:

$$\vec{x}_{n,decompressed} = \vec{x}_{n,transformed} \cdot P^T + \langle \vec{x} \rangle \tag{70}$$

On obtient donc de nouveau des vecteur de dimension D.

On voit dans cette expression en quoi c'est une expression avec perte. En effet, on a $\vec{x}_{n,decompressed} = ((\vec{x}_n - \langle \vec{x} \rangle) \cdot P) \cdot P^T + \langle \vec{x} \rangle$. L'opérateur PP^T est une matrice de dimension (D, D), mais qui intuitivement, "passe par une dimension intérmédiaire de dimension D'". Mathématiquement, on pourrait constater que PP^T n'a que D' directions indépendantes (valeurs propres distinctes).

16.4 PCA: résumé de l'algorithme

L'algorithme de la PCA est donc le suivant :

1. Calculer la matrice de covariance C (cela se fait en un temps linéaire en la taille des données) : $C = \frac{1}{N} \sum_{n=1}^{N} (\vec{x}_n - \langle \vec{x} \rangle)(\vec{x}_n - \langle \vec{x} \rangle)^T$

[8] Le sous-espace des matrices diagonalisables dans $\mathbb{C}$ est dense dans l'espace des matrices (à valeurs dans $\mathbb{C}$. Ceci n'est pas vrai dans $\mathbb{R}$, mais à part pour de données pathologiques, une matrice de covariance sera diagonalisable.

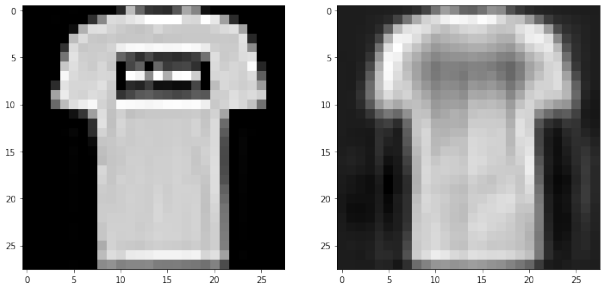


Figure 9: On passe d'une image de dimension $D = 784$ à $D' = 30$, puis on revient à l'espace d'origine pour visualisation. On a $D' = 30 \sim \sqrt{D}$ soit presque 30 fois moins, et pourtant on peut toujours reconnaître l'image.

2. Diagonaliser C. Cela se fait en un temps $O(D^3)$, a priori (voir remarque plus bas). Cela revient à chercher une matrice de passage U telle que
$$C = U\Lambda U^{-1}$$
où Λ est une matrice diagonale dont les coefficients diagonaux sont les valeurs propres de C rangées dans l'ordre décroissant.

3. Garder seulement les D' premières valeurs propres (on s'intéresse uniquement aux plus grandes valeurs propres qui permettent d'obtenir plus grande variance, que l'on cherche justement à maximiser) : $P = (\vec{u}_1, ..., \vec{u}_{D'})_{D,D'}$

4. Projeter (transformer) : $\vec{x}_{n,transformed} = (\vec{x}_n - \langle\vec{x}\rangle).P$

Par ailleurs si on ne cherche que les D' plus grandes valeurs propres, on peut le faire en un temps $O(D'D^2)$ au lieu de $O(D^3)$, ce qui est très appréciable (on part du principe que $D' \leq D$). Il existe aussi des solutions encore approximatives plus rapides avec résultats légèrement aléatoires. D'autant que dans le cas de la projection ce n'est pas grave d'avoir quelques erreurs.

Une fois cet algorithme appliqué, on peut effectuer la transformation inverse pour revenir à l'espace de départ (c'est une compression avec perte).

Attention : Quand on effectue une PCA sur une une image de dimension D, on obtient D' nombres, mais on n'a plus du tout a faire a une image, la donnée n'est plus visualisable – on peut cependant la décompresser ensuite pour revenir à une image.

16.5 Ressources

Concernant ce Chapitre, vous avez:

- (ressource bilingue)
`https://gitlab.inria.fr/flandes/ias/-/blob/master/PCA-intuitive-showcase.ipynb`
- (ressource bilingue)
`https://gitlab.inria.fr/flandes/ias/-/blob/master/PCA-proof-of-exact-computation.ipynb`
- (en anglais)
`https://plot.ly/ipython-notebooks/principal-component-analysis/`
- Bishop [**?**] chapter 12, page 561-570 (ou plus si vous êtes intéressé.e.s).

17 Modèles Bayésiens (CM7+8, séances 6+8)

Les techniques décrites ici peuvent être vues comme des *estimations de densité*.

Lorsqu'on souhaite "fitter" les paramètres d'une densité de probabilité sur un ensemble de données, la méthode la plus naturelle consiste à choisir les paramètres tels que les données soient "les plus vraisemblables" ou autrement dit, qu'elles soient "réalistes", c.a.d. qu'il soit crédible que ces données aient été générées par cette distribution, avec ces paramètres.

Cette approche s'appelle l'estimation par le Maximum de Vraisemblance (des données). On cherche ainsi à trouver les paramètres optimaux θ^* qui maximisent:

$$\theta^* = \text{argmax}_\theta(\mathbb{P}(X|\theta)) \tag{71}$$

Notation: l'exposant .* est souvent utilisé pour désigner la solution d'un problème d'optimisation.

Notation: le chapeau $\hat{u}$ est souvent utilisé pour désigner l'*estimateur* d'une quantité a priori inconnu. On lira $\hat{u}$ "u chapeau" ou bien "estimateur de u".

Notation: on dit que la variable aléatoire (v.a.) X est paramétrée par les paramètres θ.

17.1 MLE: 1 variable à 1 dimension

Prenons un exemple concret, par exemple la taille en cm d'un certain nombre N d'étudiants. On dispose de N points de données, à 1 dimension, c.a.d que ce sont des nombres réels. On suppose que les tirages sont indépendants. Si X_n est la v.a. associée au n-ième tirage (indiquée ci dessous), la v.a. des N tirages est alors la v.a. produit, notée X:

$$\forall n \in [|1, ..., N|], X_n \sim X_1 \tag{72}$$

$$X \sim (X_1, X_2, \dots, X_N) \tag{73}$$

Notations: $X \sim Y$ signifie que la v.a. X suit la même loi que la v.a. Y. L'indépendance entre deux v.a. X_1 et X_2 est notée $X_1 \perp\!\!\!\perp X_2$. Ici on a $X_i \perp\!\!\!\perp X_j, \forall i \neq j$. Ici, les virgules signalent un "ET" logique.

On dispose de données, qu'on note plutôt $X_{(N,1)}$ que X, afin de les distinguer de la variable aléatoire X. Les données sont la *réalisation* d'une certaine loi (ou bien quelque chose qu'on arrivera jamais à décrire mathématiquement, selon les problèmes). Les données $X_{(N,1)}$ sont donc une liste: $X_{(N,1)} = (x_1, x_2, \dots, x_N)$, avec $x_n \in \mathbb{R}$.

On définit alors la probabilité d'avoir obtenu le tirage $X_{(N,1)}$, depuis la v.a. X:

$$\mathbb{P}(X = X_{(N,1)}) = \mathbb{P}(X_1 = x_1, X_2 = x_2, \dots X_N = x_N) \qquad \text{(Définition, toujours vraie)} \tag{74}$$

$$= \prod_n^N \mathbb{P}(X_n = x_n) \qquad \text{(Car les } X_n \text{ sont indépendants)} \tag{75}$$

Il faut maintenant rendre les choses plus concrètes en choisissant un *modèle* mathématique, c'est-à-dire une expression **explicite** pour la forme de $\mathbb{P}(X_n = x_n)$. Ici, pour aujourd'hui, on suppose que tous les points sont distribués selon la (même) loi **Gaussienne**:

$$\mathbb{P}(X_n \in [x, x + dx]) = \rho(x)\mathrm{d}x, \tag{76}$$

$$\rho(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{77}$$

Où $\theta = (\mu, \sigma)$ sont les 2 paramètres réels à *estimer* (à "fitter", en vocabulaire de Machine Learning). Attention, de façon générale, on aurait pu faire un autre choix de représentation, à la place de la Gaussienne: on aurait pu prendre une loi de Poisson, de Bernoulli, etc. C'est un *choix* de *modélisation*.

Notation: $\rho(x)$ se lit rho de x, c'est la *densité* associée à la v.a. X, qui est une v.a. continue.

L'idée de maximiser la vraisemblance des données revient alors, concrètement, à **maximiser la vraisemblance des données sachant les paramètres**, notée $\mathcal{L}$ (comme Likelihood, Vraisemblance en anglais):

$$\mathcal{L}(\theta) = \mathbb{P}(X = X_{(N,1)}|\theta) \tag{78}$$

$$\theta^* = \text{argmax}_\theta(\mathcal{L}(\theta)) \tag{79}$$

Quel que soit le choix de modélisation, il y aura toujours au moins un paramètre à *estimer* d'après les données: c'est pour cela qu'on parle d'*estimateur par le maximum de vraisemblance* (*Maximum Likelihood Estimate*, en anglais, ou EMV en Français).

Il y a une astuce qui est appliquée quasi systématiquement: le argmax d'une fonction est aussi le argmax du log de cette fonction, car log est strictement croissante:

$$\theta^* = \text{argmax}_\theta(\mathcal{L}(\theta)) = \text{argmax}_\theta(\log(\mathcal{L}(\theta))) = \text{argmax}_\theta(\ell(\theta)) \tag{80}$$

$$\ell(\theta) = \log \mathcal{L}(\theta) \quad \text{(est nommée la log-vraisemblance ou log-likelihood)} \tag{81}$$

Le reste n'est plus qu'un bête calcul. Pour le modèle Gaussien, on calcule une bonne fois $\ell(\theta)$:

$$\ell(\theta) = \log\left(\rho(X = X_{(N,1)}|\theta)\right) \tag{82}$$

$$= \log\left(\prod_n^N \rho(X_n = x_n|\theta)\right) \tag{83}$$

$$= \log\left(\prod_n^N \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x_n-\mu)^2}{2\sigma^2}\right)\right) \tag{84}$$

$$= \sum_n^N \left(\log \frac{1}{\sqrt{2\pi\sigma^2}}\right) + \sum_n^N \left(\log\exp\left(-\frac{(x_n-\mu)^2}{2\sigma^2}\right)\right) \tag{85}$$

$$= -N\log(\sqrt{2\pi}\sigma) - \sum_n^N \frac{(x_n-\mu)^2}{2\sigma^2} \tag{86}$$

On cherche le maximum (extremum) d'une fonction en cherchant à annuler sa dérivée première (on pourra vérifier que la dérivée seconde est négative). Donc, au point idéal $\theta^* = (\mu^*, \theta^*)$, on aura:

$$0 = \frac{\partial}{\partial\mu}\ell(\theta) \tag{87}$$

$$0 = 0 - \sum_n \frac{2(x_n-\mu)}{2\sigma^2}(-1) \tag{88}$$

$$0 = \sum_n (x_n - \mu) \tag{89}$$

$$N\mu = \sum_n x_n \tag{90}$$

$$\mu = \frac{1}{N}\sum_n x_n \hat{=} \overline{x} \tag{91}$$

C'est-à-dire qu'on trouve que l'estimateur du paramètre μ par maximum de vraisemblance, noté μ^* ou $\hat{\mu}$, est égal à la moyenne empirique, notée $\overline{x}$. On a trouvé : $\hat{\mu} = \overline{x}$

De même pour σ :

$$0 = \frac{\partial}{\partial\sigma}\ell(\theta) \tag{92}$$

$$0 = -N\frac{1}{\sigma} - \sum_n \frac{2(x_n-\mu)^2}{2\sigma^3} \tag{93}$$

$$0 = \frac{N}{\sigma}\sigma^3 + \sum_n \frac{(x_n-\mu)^2}{\sigma^3}\sigma^3 \tag{94}$$

$$\sigma^2 = \frac{1}{N}\sum_n^N (x_n-\mu)^2 \tag{95}$$

On trouve ici encore que le meilleur estimateur du paramètre σ pour modéliser les données, σ^*, aussi noté $\hat{\sigma}$, est égal à l'écart-type empirique, $\sigma_{empirique}$ (là il n'y a pas vraiment de notation pour résumer l'expression $\sqrt{\frac{1}{N}\sum_n^N (x_n-\mu)^2}$).

Finalement, on retrouve les estimations fréquentistes, c.a.d que l'estimation du maximum de vraisemblance coïncide avec les estimations empiriques (par exemple, la variance empirique, c'est $\frac{1}{N}\sum_n^N (x_n-\mu)^2$) Mais la méthode est très générale !

17.2 1 variable à D dimensions

On précise d'abord la structure des données:
N, la taille des données d'apprentissage (le nombre d'exemples)
D, la dimension de chaque point de donnée
$X_{(N,D)}$: les données d'entraînement (par exemple, un ensemble d'images). $\vec{x}_n$ est l'image numéro n, elle correspond à D nombres réels (intensités de gris des pixels)
Θ: les paramètres à optimiser (le détail dépend du modèle choisi).

17.2.1 Dimensions indépendantes

Si les dimensions sont supposées indépendantes, alors c'est facile, tout se factorise. Par exemple, si on a des gaussiennes à plusieurs dimensions, mais avec chaque dimension indépendante, alors on a, pour un tirage $\vec{x}_n \in \mathbb{R}^D$, une densité:

$$\rho(\vec{x}_n) = \prod_{d=1}^{D} \frac{1}{\sqrt{2\pi\sigma_d^2}} \exp\left(-\frac{(x_{nd}-\mu_d)^2}{2\sigma_d^2}\right) \tag{96}$$

Où on a noté x_{nd} la d-ième composante du vecteur $\vec{x}_n$. Quand on prend le log, ce produit là aussi devient une somme, et lorsqu'on cherche la meilleure estimation du paramètre σ_d par exemple, on obtient simplement la même chose qu'en 1D:

$$\sigma_d^2 = \frac{1}{N}\sum_{n}^{N}(x_{nd}-\mu_d)^2 \tag{97}$$

17.2.2 Dimensions corrélées

Quand les dimensions sont corrélées, on doit caractériser leur corrélation.

Une caractérisation de la corrélation entre les différentes dimensions se fait, au premier niveau, par la matrice de covariance. On peut voir l'ensemble des tirages, $X_{(N,D)}$, comme un ensemble de tirages simultanés de lois distinctes: le n-ième tirage correspond au tirage simultané des variables aléatoires $X_{n1}, X_{n2}, \dots X_{nD}$ (Dans le rappel de cours ci dessous, on a X et Y, ici ça correspond par exemple à X_{n1}, X_{n2}).

La matrice de covariance est donc la matrice des covariances des v.a. associées à chaque dimension (on pourrait les appeler les X_d, mais c'est un abus de notation), elle est de taille (D, D). En loi, sa composante d, d' est définie par:

$$K_{d,d'} = Cov[X_d, X_{d'}] = \mathbb{E}[(X_d - \mathbb{E}[X_d])(X_{d'} - \mathbb{E}[X_{d'}])] \tag{98}$$

Son estimateur statistique est:

$$\hat{K}_{d,d'} = \frac{1}{N}\sum_{n}^{N}(x_{n,d} - \overline{X_d}(x_{n,d'} - \overline{X_{d'}})] \tag{99}$$

où $\langle X_d \rangle = \frac{1}{N}\sum_n^N x_{n,d}$ dénote la moyenne empirique (qui se trouve être l'estimateur de l'espérance, ce n'est pas un hasard).

La diagonale de cette matrice sera constituée par la liste des variances de chaque dimension.

17.2.3 Caractérisation des corrélations: la Covariance (rappel du poly de maths de L2)

Lorsque deux v.a. X, Y sont corrélées (non indépendantes), la loi de leur somme, $Z = X + Y$, n'est pas déterminée de façon simple à partir des lois de chacune. En particulier, les moments d'ordre k de Z ne sont pas toujours égaux à la somme des moments d'ordre k de X et Y.
Le moment d'ordre zéro est toujours égal à 1, c'est la normalisation: $\mathbb{E}[Z^0] = 1 \neq \mathbb{E}[X^0] + \mathbb{E}[Y^0] = 1 + 1$.
Le premier moment, lui, est linéaire: $\mathbb{E}[X+Y] = \mathbb{E}[X] + \mathbb{E}[Y]$ (que X, Y soient indépendantes ou pas !).
Le second moment de $Z = X + Y$ est **une première façon de caractériser la corrélation** entre X et Y.
On définit ainsi la **covariance** de X, Y:

$$Cov(X,Y) = \mathbb{E}\Big[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])\Big] \tag{100}$$

Moralement, la covariance correspond au produit de X et Y: on peut retenir que, en gros, $Cov(X,Y) \approx \mathbb{E}[XY]$. En gros seulement, car il faut d'abord enlever le premier moment pour que cette égalité fonctionne réellement. On a soustrait les espérances de chaque v.a. afin d'enlever la partie "triviale" (constante) des deux variables. Et pourquoi vouloir calculer le produit ? Parce que c'est l'opération la plus simple, qui soit tout de même non triviale (or l'addition est triviale, puisque le premier moment est linéaire, lui).

Cet aspect générique est également apparent par le fait que la covariance apparait dans $V[Z] = V[X+Y]$, soit le deuxième moment de la somme.

$$V[X+Y] = V[X] + V[Y] + 2\ Cov(X,Y) \tag{101}$$

Propriétés:

- $X \perp\!\!\!\perp Y \implies Cov(X,Y) = 0$
- $Cov(X,Y) = 0 \not\implies X \perp\!\!\!\perp Y$
- $Cov(X,X) = V[X]$
- $Cov(X,Y) = Cov(Y,X)$
- $Cov(aX,Y) = aCov(X,Y)$
- $Cov(X+a,Y) = Cov(X,Y)$
- $Cov(X+Y,W) = Cov(X,W) + Cov(Y,W)$

Coefficient de corrélation: Si le second moment de X est fini, et le second moment de Y aussi ($V[X] < \infty, V[Y] < \infty$), le **coefficient de corrélation** de X, Y est :

$$Corr(X,Y) = \frac{Cov(X,Y)}{\sigma[X]\sigma[Y]} \tag{102}$$

On remarque que même si X, Y ont chacun des unités (par exemple des cm et des kg), le coefficient de corrélation est **adimensionné** (il n'a pas d'unités).

Le coeff. de corrélation varie entre -1 (anti corrélation complète, on peut essayer de prouver que $X = -Y$ en loi) et 1 (corrélation complète, on peut essayer de prouver que $X = Y$ en loi). Mais attention, si il vaut 0, cela ne signifie pas nécéssairement que $X \perp\!\!\!\perp Y$!

17.2.4 Cas notable: Gaussienne multi-variée

Le cas de la Gaussienne multi-variée est un cas de variable aléatoire dotée d'une covariance fixée qui revêt un intérêt particulier, car dans ce cas, la covariance caractérise parfaitement les corrélations. Sa densité s'écrit:

$$f_{\boldsymbol{\mu},\boldsymbol{\Sigma}}(\boldsymbol{x}) = \frac{1}{(2\pi)^{N/2}\left|\boldsymbol{\Sigma}\right|^{1/2}}\ \exp\left[-\frac{1}{2}\left(\boldsymbol{x}-\boldsymbol{\mu}\right)^{\top}\boldsymbol{\Sigma}^{-1}\left(\boldsymbol{x}-\boldsymbol{\mu}\right)\right] \tag{103}$$

Ici, pour changer, on a adopté la notation plus mathématicienne/informaticienne (moins explicite, plus compacte): les vecteurs sont en gras, les matrices aussi. Ici $\boldsymbol{\Sigma}$ est la matrice de covariance de la variable aléatoire associée aux $\boldsymbol{x}$, elle est de dimension (D, D). On note qu'on a besoin de $\boldsymbol{\Sigma}^{-1}$, il faut donc qu'elle soit inversible, et par ailleurs la notation $|\boldsymbol{\Sigma}| = det\boldsymbol{\Sigma}$ indique le déterminant (qui doit être non nul).

Allez jouer avec le script "exemples-Matrices-de-Covariance+Genereration-de-Gaussiennes" pour voir quelques gaussiennes multi-variées en 2D, déjà. (en gros, il suffit d'utiliser la méthode "X = np.random.multivariate_normal(mu, cov, (N))", où mu est le vecteur des moyennes, et cov la matrice de covariance des données souhaitée.

17.3 K variables à D dimensions

On peut supposer que les données sont issues de différentes sources (des variables aléatoires), c.a.d. qu'elles sont un mélange de différentes lois de probabilités. Ce cas est plus complexe. On peut d'abord traiter le cas, très simple, où chaque source produit des données X, mais qu'on détient en même temps le label y ou k qui permet d'identifier la source. C'est le cas ci dessous.

17.4 K variables à D dimensions, avec label connu: le Modèle Bayésien Naïf

Une fois qu'on a compris l'idée du calcul du MLE (Maximum Likelihood Estimate), on peut en faire un modèle d'apprentissage supervisé, très facilement. C'est un modèle un peu pauvre, mais qui est la brique de base pour des modèles plus complexes, plus expressifs. Il est donc important de bien comprendre cette brique de base.

Par ailleurs, cela fait un bon exercice sur la manipulation d'indices et de sommes ou produits assez simples.

Ici, pour être concret, on choisira un modèle de Bernoulli. On rappelle que la distribution de proba de Bernoulli peut s'écrire: $\mathbb{P}(X=1)=p, \mathbb{P}(X=0)=1-p$, ce qui peut se résumer sous la forme plus dense: $\mathbb{P}(X=x)=p^x(1-p)^{1-x}$. (on utilise le fait que x vaut forcément 0 ou 1, et que $A^0=1, \forall A$).

17.4.1 Définition du problème, et structure des données

On peut imaginer qu'on s'intéresse à de la classification supervisée d'images en noir et blanc.

On précise la structure des données: N, la taille des données d'apprentissage.
D, la dimension des données (= nombre de pixels)
K, le nombre de classes présentes dans les données à classifier
$X_{(N,D)}$: les données d'entraînement (typiquement, un ensemble d'images). $\vec{x}_n$ est l'image numéro n, elle correspond à D nombres réels (intensités de gris de pixels)
$y_{(N)}$: les classes des données ($y[n]$ est la classe de la donnée numéro n (typiquement, à valeurs dans $[1, ..., K]$)
Θ: les paramètres à optimiser (le détail dépend du modèle choisi, mais il y a au moins K paramètres, probablement KD ou plus.

17.4.2 Apprentissage des paramètres du modèle

On souhaite maximiser la vraisemblance de données, pour un modèle choisi, c.a.d. pour une forme de $\mathbb{P}(X=(\vec{x})_n|y=k,\Theta)$ fixée (par exemple, modèle Gaussien ou de Bernoulli, ou une autre forme explicite de la relation entre $\mathbb{P}(X=(\vec{x})_n|y=k,\Theta)$ et les variables $x_{n,d}, y, \Theta_{k,d}$. Intuitivement, le terme de vraisemblance peut être remplacé par crédibilité, degré de réalisme: on se demande: "quels sont les paramètres Θ qui font qu'observer telle ou telle donnée (image) est crédible/réaliste ?". C'est donc assez raisonnable de maximiser cette chose là.

Maximiser la vraisemblance de données revient à apprendre les valeurs des paramètres Θ telles que $\mathcal{L}(\Theta)=\mathbb{P}(X=(\vec{x})_n|y=k,\Theta)$ soit maximale. On peut appeler Θ^* la solution de ce problème d'optimisation:

$$\Theta^* = argmax_\Theta(\mathcal{L}(\Theta)) \tag{104}$$

On développe cette expression, dans le but d'obtenir une forme explicite dont la dérivée ne soit pas trop dure à calculer, et par indépendance des variables aléatoires $X_n, n \in [\![1, N]\!]$, on a:

$$\begin{aligned}
\Theta^* &= argmax_\Theta\left(\log(\mathcal{L}(\Theta))\right) && (105)\\
&= argmax_\Theta\left(\log \mathbb{P}(X=(\vec{x})_n|y_{(n)},\Theta)\right) && (106)\\
&= argmax_\Theta\left(\log \prod_n^N \mathbb{P}(X_n=\vec{x}_n|y_n=k,\Theta)\right) && (107)\\
&= argmax_\Theta\left(\sum_n^N \log \mathbb{P}(X_n=\vec{x}_n|y_n=k,\Theta)\right) && (108)
\end{aligned}$$

À ce stade on est bien obligés de choisir une forme explicite pour l'expression un peu abstraite $\mathbb{P}(X_n=\vec{x}_n|y=k,\Theta)$. C'est ce qu'on appelle un *modèle* des données.

Les modèles Bayésiens **naïfs** correspondent à supposer que les features d'entrée sont **indépendantes** ! C'est une énorme simplifcation ! Dans la vraie vie, dans les données que l'ont rencontre typiquement, il y a des corrélations entre les features (c'est ce qu'on voit par exemple avec la PCA). Concrètement, l'indépendance des features signifie que pour chaque sample $\vec{x}_n$ (point de donnée), chaque valeur $(x_n)_d$ est indépendante des autres (des autres $(x_n)_{d'}$, c.a.d. des autres pixels de la même image). On écrit donc:

$$\mathbb{P}(X_n=\vec{x}_n|y_n=k,\Theta) = \prod_d^D \mathbb{P}(X_{n,d}=x_{n,d}|y_n=k,\Theta) \tag{109}$$

Il y a ici un petit abus de notation: on écrit $X_{n,d} = x_{n,d}$, le terme de gauche est la variable aléatoire $X_{n,d}$, le terme de droite est la valeur prise dans la réalisation de cette v.a., que l'on nomme aussi (c'est un léger abus), $x_{n,d}$.

On note $\mathbb{1}_{y_n=k}$ la fonction qui vaut 1 quand $y_n = k$ et 0 le reste du temps. C'est un moyen rapide d'imposer $k = y_n$ partout dans l'équation.

Dans le cas où le modèle de la distribution des features d'entrée est choisi Bernoulli, pour chaque feature, cela s'écrit, plus concrètement: $\mathbb{P}(X_n = \vec{x}_n | y_n = k, \Theta) = \prod_d^D p_{k,d}^{x_{n,d}} (1 - p_{k,d})^{(1-x_{n,d})} \mathbb{1}_{y_n=k}$. Ici les paramètres Θ sont, concrètement, les $p_{k,d}$.

$$\Theta^* = argmax_\Theta \left(\sum_n^N \log \Big(\prod_d^D p_{k,d}^{x_{n,d}} (1 - p_{k,d})^{(1-x_{n,d})} \Big) \mathbb{1}_{y_n=k} \right) \tag{110}$$

$$= argmax_\Theta \left(\sum_n^N \sum_d^D \log \Big(p_{k,d}^{x_{n,d}} (1 - p_{k,d})^{(1-x_{n,d})} \Big) \mathbb{1}_{y_n=k} \right) \tag{111}$$

$$= argmax_\Theta \left(\sum_n^N \sum_d^D \left[x_{n,d} \log(p_{k,d}) + (1 - x_{n,d}) \log(1 - p_{k,d}) \right] \mathbb{1}_{y_n=k} \right) \tag{112}$$

Pour trouver le maximum de cette fonction en Θ, on dérive par rapport à Θ (ici, concrètement, $p_{k,d}$) et on cherche la valeur de $p_{k,d}$ telle que la dérivée soit nulle:

$$0 = \frac{\partial \log \mathcal{L}(\Theta)}{\partial p_{k,d}} \tag{113}$$

$$= \frac{\partial}{\partial p_{k,d}} \Big(\sum_n^N \sum_{d'=1}^D \left[x_{n,d'} \log(p_{k,d'}) + (1 - x_{n,d'}) \log(1 - p_{k,d'}) \right] \mathbb{1}_{y_n=k} \Big) \tag{114}$$

Or, la dérivée s'annule pour tout $d' \neq d$, donc:

$$0 = \sum_n^N \left[\frac{x_{n,d}}{p_{k,d}} - \frac{1 - x_{n,d}}{1 - p_{k,d}} \right] \mathbb{1}_{y_n=k} + \sum_{d' \neq d}^D (0) \tag{115}$$

$$0 = \sum_n^N \left[x_{n,d}(1 - p_{k,d}) - (1 - x_{n,d}) p_{k,d} \right] \mathbb{1}_{y_n=k} \tag{116}$$

$$= \sum_n^N \left[x_{n,d} - p_{k,d} \right] \mathbb{1}_{y_n=k} \tag{117}$$

Ce qui donne, en distribuant la somme:

$$\sum_n^N p_{k,d} \mathbb{1}_{y_n=k} = \sum_n^N x_{n,d} \mathbb{1}_{y_n=k} \tag{118}$$

$$p_{k,d} \sum_n^N \mathbb{1}_{y_n=k} = \sum_n^N x_{n,d} \mathbb{1}_{y_n=k} \tag{119}$$

$$p_{k,d} = \frac{\sum_n^N x_{n,d} \mathbb{1}_{y_n=k}}{\sum_n^N \mathbb{1}_{y_n=k}} \tag{120}$$

Ce qui correspond à faire la moyenne des pixels numéro d des images de classe k: en effet le dénominateur est simplement une façon de compter le nombre d'images dans la classe k.

On peut remarquer que dans ce cas présent, on a pu faire les calculs exactement jusqu'au bout, et donc l'apprentissage se fait d'un coup, il n'y a pas d'itérations.

On peut aussi remarquer que le résultat final est extrêmement simple: on fait simplement la moyenne empirique du pixel numéro d de toutes les images étant de la classe k, et cela nous donne le paramètre p_{kd} associé à ce pixel. Dans le cas où il y aurait des corrélations entre les pixels, modélisées par des lois choisies, le calcul peut vite devenir beaucoup plus complexe, voire infaisable analytiquement.

17.4.3 Fonction de décision

Dans cette partie, on va utiliser la formule de Bayes, dans une forme un peu généralisée:

$$\mathbb{P}(A|B,C)\mathbb{P}(B|C) = \mathbb{P}(A,B|C) \tag{121}$$

Pour un nouveau point de donnée (données de validation ou test), on connaît $\vec{x}$ et on souhaite prédire y. On connaît désormais les paramètres Θ. On choisit la classe qui maximise la vraisemblance des données.

Intuitivement, nous choisissons la classe k comme celle qui correspond le plus probablement à l'image observée $\vec{x}$, en supposant qu'elle soit générée par les paramètres que nous avons appris (le vecteur (p_k), de dimensions D).

Mathématiquement, cela correspond à:

$$\begin{aligned} k^* &= argmax_k(\mathbb{P}(y=k|\vec{x},\Theta)) && (122)\\ &= argmax_k\left(\frac{\mathbb{P}(y=k\cap\vec{x}|\Theta)}{\mathbb{P}(\vec{x}|\Theta)}\right) && (123)\\ &= argmax_k\left(\frac{\mathbb{P}(\vec{x}|y=k,\Theta)\mathbb{P}(y=k|\Theta)}{\mathbb{P}(\vec{x}|\Theta)}\right) && (124)\\ &= argmax_k\mathbb{P}(\vec{x}|y=k,\Theta)\mathbb{P}(y=k|\Theta) && (125)\end{aligned}$$

La dernière ligne s'obtient en remarquant que le dénominateur ne dépend pas de k, donc il n'a pas d'importance pour déterminer l'emplacement du maximum.

Le terme $\mathbb{P}(\vec{x}|y=k,\Theta)$ a déjà été vu plus haut, il dépend du modèle choisi (Bernoulli Naïf, Gaussien Naïf ou autre).

Le terme $\mathbb{P}(y=k|\Theta)$ correspond à la probabilité d'observer une image de classe k, connaissant les paramètres appris (Θ) mais sans prendre connaissance de l'image, $\vec{x}$. C'est à dire en fait la probabilité qu'une image tirée au hasard soit de la classe k, parmi les images de *l'ensemble d'apprentissage*. C'est donc une quantité (K nombre réels) qu'il faut aussi *apprendre* la fréquence d'apparition de la classe k:

$$\pi_k = P_k = \mathbb{P}(y=k|\Theta) = \frac{\sum_n^N \mathbb{1}_{y_n=k}}{\sum_n^N 1} \tag{126}$$

On voit que cet "apprentissage" se fait d'un coup, sans itérations. C'est simplement le calcul de la fréquence de la classe k dans le training set. Cette quantité est souvent notée π_k ou P_k.

Concrètement, on a donc:

$$k^* = argmax_k\mathbb{P}(\vec{x}|y=k,\Theta)\mathbb{P}(y=k|\Theta) = argmax_k\Big(\prod_{d=1}^{D} p_{k,d}^{x_d}(1-p_{k,d})^{(1-x_d)}\frac{\sum_n^N \mathbb{1}_{y_n=k}}{N}\Big) \tag{127}$$

Numériquement, il est préférable de maximiser le log de ce nombre, pour éviter les erreurs d'arrondis. Il faudra cependant penser à prendre garde à ce que aucun des $p_{k,d}$ ne soit nul. Si tel est le cas (et ce sera très probablement le cas, dans MNIST ou ce genre de dataset très simple), on doit corriger ce problème numérique, en remplacant:

$$\begin{aligned} k^* &= \text{argmax}_k\left(\log\prod_{d=1}^{D}\left[p_{k,d}^{x_d}(1-p_{k,d})^{(1-x_d)}\right] + \log\frac{\sum_n^N \mathbb{1}_{y_n=k}}{N}\right) && (128)\\ &= \text{argmax}_k\left(\sum_{d=1}^{D}\left[(x_d)\log(p_{k,d}) + (1-x_d)\log(1-p_{k,d})\right] + \log\pi_k\right) && (129)\end{aligned}$$

par

$$k^* = \text{argmax}_k\left(\sum_{d=1}^{D}\left[(x_d)\log(p_{k,d}+\varepsilon) + (1-x_d)\log(1-p_{k,d}+\varepsilon)\right] + \log\pi_k\right) \tag{130}$$

Avec $\varepsilon = 10^{-8}$ par exemple.

17.4.4 Pre-processing

Ici, on a utilisé un modèle de Bernoulli. Implicitement, cela suppose que les entrées sont à valeur dans l'ensemble $\{0, 1\}$. Si on veut appliquer cet algo sur des données (images en noir et blanc), que faudra-t il faire comme pré-processing ?

17.4.5 Prior

Imaginez qu'on vous donne un ensemble d'apprentissage déséquilbré, où une classe est surreprésentée. Supposez que vous savez que dans les données, en général (donc, dans l'ensemble de test, en particulier), les données sont réparties de manière égale entre toutes les classes. Que pouvez-vous changer dans la fonction de décision, pour en tenir compte ? Il s'agit d'un exemple très simple de *prior*.

17.4.6 Modèle Gaussien Naïf

Exercice: Calculer les équations d'apprentissage pour le modèle gaussien naïf.

17.4.7 Modèle Gaussien, mais pas si naïf

En gros: et si on mettait un modèle avec une matrice de covariance non diagnonale entre les features ?

18 Autres modèles: quelques intuitions sur des classiques (CM9, séance 9)

18.1 Arbres de décision

L'idée d'un arbre de décision est de construire un arbre de questions successives. Il s'agit de classifier nos données features par features.

Chaque branche de l'arbre comprend une question sur une feature et un seuil qui engendrent un branchement et une division des données. La génération de l'arbre est la suivante, les questions de chaque sous groupes sont différentes et on questionne seulement les données restantes dans la branche.

Par exemple sur le dataset Iris : est ce que le pétale a une largeur ≤ 0.8 ? Si oui, on "descend" les données correspondantes à gauche et les autres a droite.

Il est à noter que l'on peut poser plusieurs fois d'affilé des questions différentes à propos de la même feature (par exemple dans le cas où il y a trois classes ou plus).

Lorsqu'une feuille contient des exemples (échantillons) de plusieurs classes différentes, on peut recourir à plusieurs stratégies.

- Voter à la majorité simple, c'est-à-dire choisir la classe dont les exemples représentent plus de 50% des échantillons présents dans la feuille (en cas d'égalité, utiliser l'aléatoire ou un autre critère)
- Diviser encore les données et descendre jusqu'à ce que chaque feuille de l'arbre soit pure (qu'il ne reste plus que des représentants d'une seule classe dedans)

On souhaiterait construire notre arbre de décision de manière à n'obtenir que des feuilles pures (ou mono-classe). A chaque génération, il faut donc choisir :

1. Quelle feature sera le sujet de la question (quel indice $d \in [1, D]$ choisir, avec D la dimension des données)
2. Quelle sera la valeur du seuil. Ces deux choix constituerons la question, qui peut se poser sous la forme "feature #d $<$ seuil ?".

On procédera donc comme suit : Supposons la feature d fixée et connue.On choisit le seuil tel que la pureté des feuilles obtenues via ce seuil soit maximale. Pour juger de cette pureté, on peut se baser les critère de Gini ou d'Entropie (ou de MSE en régression). La formulation mathématique exacte de la pureté peut être retrouvée dans la documentation de sklearn :

https://scikit-learn.org/stable/modules/tree.html#mathematical-formulation

Voici un algorithme pour construire un bon arbre de décision :

Algorithm 1 Construction d'un arbre

```
procedure CONSTRUCTION D'UN ARBRE(X, MaxIter)
    for chaque feature do
        for chaque seuil possible do
            Calculer la pureté de la division possible
    Prendre la meilleure combinaison        ▷ Meilleure feature avec son meilleur seuil
    Diviser les données selon cette coupure
```

On remarque que cet algorithme propose la meilleure découpe mais est très coûteux en ressources et notamment en temps. Pour pallier à ce problème, on peut se restreindre à un sous ensemble de features bien choisies. On peut donc mieux analyser les entêtes des fonctions d'arbre de décision sklearn :

```
class sklearn.tree.DecisionTreeClassifier(*, criterion='gini', splitter='best',
max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0,
min_impurity_split=None, class_weight=None, ccp_alpha=0.0)
```

```
class sklearn.tree.DecisionTreeClassifier(*, criterion='gini', splitter='best',
max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0,
min_impurity_split=None, class_weight=None, ccp_alpha=0.0)
```

Part III
Suppléments de cours - notions hors programme

Ces notions ont été vues en 2020-21, mais ne seront pas abordées cette année.

18.1.1 Au delà des features maps: les Kernels

On ne rentre pas dans le détail, mais en gros, **certaines expansions correspondent à une expansion dans un espace de dimension infinie ... plutôt cool non?** Concrètement, quand on y regarde bien, beaucoup de méthodes de ML font intervenir des produits scalaires du genre $\vec{x} \cdot \vec{x}'$ (aussi noté $(\vec{x}')^T \vec{x}$). **Ces produits scalaires sont une mesure de la proximité entre les vecteurs** x, x' (sont ils presque colinéaires, ou plutôt orthogonaux?). C'est le cas dans la PCA (dans toutes les méthodes ou la matrice de covariance apparaît, donc aussi la régression linéaire par exemple), dans les SVM (il faut bien regarder pour le voir), etc.

L'idée des méthodes à Noyau (*Kernel* en anglais, de l'allemand *Kern*, c'est devenu du français: on note bien Ker le noyau) est de substituer la distance naturelle entre deux vecteurs, $\vec{x} \cdot \vec{x}'$, par une autre fonction, plus complexe:

$$\vec{x} \cdot \vec{x}' \mapsto K(\vec{x}, \vec{x}') \tag{131}$$

On voit donc que **les features maps sont un cas particulier de Kernels, c'est-à-dire ceux qui se factorisent**: $K(x, x') = \phi(x)\phi(x')$. Lorsque ça ne factorise pas, c'est "un vrai Kernel" (pas une feature map). Toute fonction de deux variable n'est pas un Kernel: il faut respecter la condition de Mercer.. en gros, être une bonne métrique dans l'espace considéré: par exemple, être un opérateur semi-défini positif.. bref.

Le "vrai" Kernel classique est le RBF (*Radial Basis Function*).

$$K_{RBF}(x, x') = \exp\left(-\gamma ||x - x'||_2^2\right) \tag{132}$$

Il n'y a pas de forme factorisée finie. Une forme factorisée (mais de dimension infinie) est:

$$\phi_{RBF}(x) = exp(-\gamma x^2) \left[1, \sqrt{\frac{2\gamma}{1!}}x, \sqrt{\frac{(2\gamma)^2}{2!}}x^2, \sqrt{\frac{(2\gamma)^3}{3!}}x^3, \dots\right] \tag{133}$$

D'où l'affirmation que les noyaux correspondent à des expansions de dimension inifine ! On peut remarquer que ce Kernel est invariant par translation: $K_{RBF}(x, x') = f(x - x')$, on ne dépend que de la distance entre x et x', pas de leur position absolue.

18.1.2 Aspects computationels

Les features peuvent etre calculées une fois pour toutes.

Les Kernels sont calculés à la volée. Ça peut vite être très couteux.

19 [Hors programme/ Optionnel] Prélude au modèles Bayésiens: Estimation de Densité

Ici, on fait de l'apprentissage non supervisé.

L'estimation de densité est l'un des cas concrets de l'apprentissage non supervisé. Il s'agit de former un *modèle* ou une *représentation* des données X, ou plus précisément, d'estimer leur densité $P(X)$, ou $\rho(x)$ (se lit "rho"). On considère un ensemble de données X, chacune représentée par un vecteur $\vec{x}$: $X = (\overrightarrow{x_n})_{n=1...N}$ où $(\overrightarrow{x_n})$ est appelé échantillon (sample), et souvent tel que $(\overrightarrow{x}_n) \in \mathbb{R}^d$ (mais parfois aussi dans $\mathbb{N}$).

19.1 Méthodes sur grilles (avec perte)

19.1.1 Méthode 0 : Histogrammes avec bins carrés

Principe de l'histogramme :
Ici, D correspond à la dimension et N au nombre de données (en rouge sur Fig. 1). Il y a 6 bins, soit les

a

Figure 10: Histogramme. N=11 points de données sont contenus dans ls 6 bins, qui couvrent l'intervalle [0,6].

intervalles $[i, i+1]$, $i \in [\![0, 5]\!]$. Pour tracer l'histogramme, on oublie les valeurs exactes, on s'intéresse seulement au nombre de données comprises dans chaque intervalle.

Pour des données en dimension D, le fait de représenter les données (on parle de représentation, même si il n'est pas possible de faire un dessin dès que $D > 3$) par un histogramme, nécessite de créer une *grille* de *cases* (les "bins").
Pour chaque case de cette grille à D dimensions, on compte le nombre de points qui sont dedans, et ce nombre (divisé par le volume de la case) est la densité estimée en cet endroit de l'espace. Cette densité est "la hauteur des barres de l'histogramme".

Remarques sur les coûts (temps, mémoire) de cette méthode:

- Le stockage en mémoire d'un histogramme, avec un nombre de bins $Nbins$ le long de chacune des D dimensions, prend une place $Nbins^D$.
- Le temps de calcul de cet histogramme (hormis le temps d'initialiser un tableau de zéros de taille $Nbins^D$), est linéaire en N: c'est $O(N)$.
- Le temps d'accès à la densité estimée par cette *représentation* des données est d'ordre 1 (on note $O(1)$), car il suffit d'accéder à la valeur du tableau.

Remarque (détail): parfois on prend des cases de taille variable (par exemple pour l'histogramme de dsitributions en loi de puissance, comme la loi de Pareto). Dans ce cas il ne faut pas oublier de diviser le nombre de points comptés par case, par le volume de la case. Les temps de calculs sont un peu rallongés, mais si on s'y prend bien, avec un peu de book-keeping, on s'en sort avec les même ordres de grandeur de coût (au moins en théorie, ça ne change pas).

19.1.2 Méthode 1 : Idée naïve

Une idée naïve pour représenter la densité serait de s'inspirer de la méthode de l'histogramme mais en remplaçant les intervalles rigides (les bins) par une autre fonction.
On évalue cependant encore la densité sur une grille régulière. Concrètement, pour une case centrée sur le point $\vec{b}$, chaque point $\vec{x}$ contribue à la "hauteur de la barre d'histogramme" ou à la "hauteur de la case", avec un poids $K(\vec{x} - \vec{b})$. (On note K car en allemand on dit Kernel pour noyau, et les allemands sont forts en maths). Ensuite, on divise par le volume de la case, et on attribue cette densité (nombre de points par unité de volume) à tous les points de cette case.
Par exemple, si on prend la fonction "tophat", qui est en fait le carré de base, qui fait penser à l'histogramme classique, c'est $K(\vec{x} - \vec{b}) = 1 - Y(|\vec{x} - \vec{b}| - b/2)$, où Y(a)=1 si $a \geq 0$ et $Y(a) = 0$ si $a < 0$ est la fonction de Heaviside ou fonction "step"). Dès qu'un point x est éloigné du point $\vec{b}$ d'une distance plus que $b/2$, ce point ne contribuera pas à la densité pour cette case là (la case centrée sur la position $\vec{b}$). Remarque: ça s'appelle tophat parce que ça ressemble à un chapeau haut de forme (tophat).
Autre exemple: on peut prendre la fonction Gaussienne, $K(x - b) = f(x - b)$.

Définition IMPORTANTE (la Gaussienne est tellement fréquente qu'il faut retenir sa formule): La fonction de densité d'une gaussienne est définie comme suit :

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

où σ^2 représente la variance de la gaussienne et μ son espérance (σ peut être vue comme la "largeur" de la gaussienne).

Par exemple, si on choisit le noyau gaussien, lorsqu'un point de données se situe entre deux gaussiennes, il est à prendre en compte pour les deux gaussiennes. Ainsi, les distances par rapport au centre de la gaussienne influent sur la hauteur.

Remarques sur les coûts (temps, mémoire) de cette méthode: c'est très similaire au cas de l'histogramme.

- Le stockage en mémoire est le même, $O(Nbins^D)$
- Le temps de calcul dépend du type de noyau choisi. Si son support (là où il est non nul, en gros) est fini, mettons qu'il fasse une taille de k fois un bin, alors on passe d'un temps de calcul linéaire (cas

a

Figure 11: 2 gaussiennes. Les deux tracés continus correspondent aux densités (“théoriques”) dont on tire des points de donnés. Des points de données ont été tirés d’après cette loi. L’histogramme des données n’a pas été tracé, mais ça ferait quelque chose qui ressemble un peu aux lois “théoriques” (plus on tirera de points, plus ça ressemblera)

histogramme) à $O(Nk^D)$, car il faut aller ajouter de la hauteur de barre aux $\sim k^D$ carrés voisins du point de donnée. Dans le cas (pire) où le noyau est de support infini (par exemple une vraie Gaussienne n’est jamais 0, même très loin du centre de la Gaussienne, ça fait toujours un peu de valeur > 0), alors chaque point contribue à chaque case de la grille. Le temps de calcul est alors de $Nbins^D.N$, ce qui est complètement énorme/horrible!

- Le temps d’accès à la densité estimée est la même, $O(1)$.

a

Figure 12: Différence entre histogrammes (à gauche) et estimation par noyau, avec noyau “tophat” (à droite). En fait le dessin de droite semble faux, mais l’idée y est.

19.2 Méthodes sans perte

19.2.1 Méthode 2 : Idée de l’estimation par noyau

Une alternative à la méthode 1 est de faire en sorte que chaque point de donnée contribue à la densité totale par ajout d’une “petite bosse”, mais sans se fixer sur une grille régulière. Chaque point de donnée pèse alors comme une fonction régulière (lisse) qu’on a choisi (c’est le noyau, $K(x)$). On peut alors évaluer la densité en n’importe quel point. Concrètement, pour un point $\vec{b}$ quelconque de l’espace, la densité estimée en ce point est

$$\hat{\rho}(\vec{b}) = \frac{1}{N}\sum_{n}^{N} K(\vec{x} - \vec{b}) \tag{134}$$

Où on supposera que le noyau est bien normalisé, c.a.d que son intégrale sur tous l’espace fait bien 1: $\int_{\mathbb{R}^D} \mathrm{d}x^D K(\vec{x}) = 1$. Si le noyau est de portée finie (de support fini, comme on a dit plus haut), alors on peut se passer de sommer sur tous les points du dataset, et essayer de juste prendre en compte ceux qui sont assez proches pour contribuer.

NB: Puisque je triche un peu, la 2ème gaussienne n’est pas totalement exacte mais c’est l’idée. Dans le cas Gaussien il y a des symétries, qui font que l’estimation par la méthode 1 coïncide avec celle de la méthode 2 (estimation par noyau), lorsqu’on regarde exclusivement le point centrale de la grille (de la méthode 1). Cependant, cette équivalence entre l’estimation par histogramme et par noyau ne fonctionne pas toujours, donc ce n’est pas très important.

Cette méthode permet de faire une estimation de densité continue (selon le caractère plus ou moins lisse du noyau choisi), ce qui n’était pas le cas des méthodes précédentes, où la valeur ne changeait que quand on passait d’une case à l’autre, sur la grille.

Cette méthode correspond à ne faire aucun calcul, en fait: on ne calcule que lorsqu’on souhaite évaluer la densité en un point de l’espace ($\vec{b}$ dans nos notations).

Remarques sur les coûts (temps, mémoire) de cette méthode: c’est très **différent** du cas de l’histogramme.

- Le stockage en mémoire correspond juste à stocker les données, donc c’est $O(N)$.

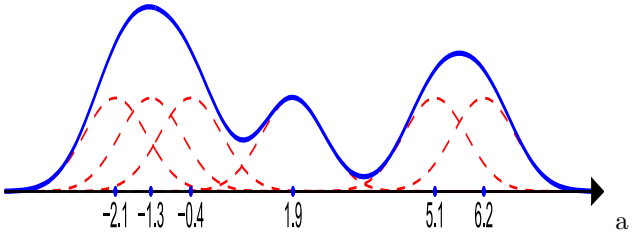


Figure 13: Illustrations de la méthode avec noyau

- Le temps de calcul est $O(1)$, car en fait on ne calcule rien.
- Le temps d'accès à la densité estimée est le temps de calcul de la formule ci dessus, donc dans le cas général, $O(N)$, où $O(N.D)$ si on est pointilleux.

19.2.2 La cumulative (exacte) - remarque

Une dernière possibilité serait d'utiliser la fonction cumulative (CDF pour Cumulative Density Function), qui permet de représenter les données sans l'aide de bins ou d'une fonction à noyau. Il s'agit d'une représentation sans perte.

Voir le TP1 pour les détails (il y a la déf exacte).

L'idée est que là encore, on ne compresse pas, on garde la connaissance totale des données. Il n'y a pas de choix arbitraire à faire, comme le choix d'une taille de bin (taille des cases de la grille). Et on peut quand même voir à quoi ça ressemble, tant que on est en $D = 1$ ou $D = 2$. Mais ça n'est pas très parlant, visuellement.

19.3 Conclusion: Comparaison des méthodes

Cette comparaison a été bien expliquée dans le corrigé du TP1, "TP1-Histogrammes-correction.ipynb".

En résumé, ce qu'il faut retenir, c'est que:

- L'intuition tend à faire penser que la représentation de N données par un histogramme avec $Nbins$ le long de chaque dimension, sera une compression des données, qui économisera de la place en mémoire, pour un petit coût en temps de calcul
- Mais en fait, en grande dimension (et même, pas si grande, voir l'exemple du TP), ça n'est pas rentable.
- Les représentations basées sur les données (estimation par noyau = méthode 2, ou bien CDF), qui coûtent 0 en calcul au départ, mais qui prennent un temps $O(N)$ à chaque fois qu'on veut consulter la densité en un point, son finalement relativement "économiques" (enfin $O(N)$ c'est pas terrible quand même).
- Conclusion: tout ça n'est pas très satisfaisant, on aimerait bien *représenter les données* (=avoir un estimateur de leur densité) pour moins cher que ça.

20 [Hors programme] Traitement automatique des langues

Ces notions ont été vues en 2020-21, mais ne sont pas au programme de l'année 2021-22.

20.1 Philosophie générale des plongements vectoriels

Sur cette première partie de cours, nous allons nous intéresser à un procédé de réduction de dimension, les **plongements**, et à leur visualisation grâce à la PCA. En effet on peut apercevoir des nuages de points de grande dimension grâce à la PCA (en projettant sur les 2 premières composantes).

20.1.1 Passage d'un texte brut à une représentation mathématique

On s'intéresse à la façon de passer d'un texte brut à quelque chose qui se calcule mathématiquement. On veut l'appliquer d'un "point de vue statistique".

On constate qu'aujourd'hui, la plupart des textes sont numérisés .
Tout texte numérisé qu'on a est une suite de caractères, qui sont codés peu importe le système d'écriture, dans le code Unicode par exemple. Tous ces caractères représentent plusieurs dizaines de milliers d'entrée dont 140k rien que pour l'Unicode, une autre norme est le code ASCII qui lui n'en contient que 128. Le chinois demande 40k et plus caractères, et dans l'Unicode, on a aussi le code des caractères égyptien. Aujourd'hui tout ce qui est lisible est numérisable.

Les correcteurs orthographiques, dans les traitements de textes, calculent déjà des distances entre les mots tapés et ceux d'un dictionnaire.. etc.

Il y a un lien entre son et prononciation des lettres/caractères. Cependant le lien entre le son et le sens semble arbitraire, mais des exceptions existent. En effet, il existe une sexualisation des prénoms, a et i pour les filles et o, u, ou pour les garçons.

Sur wikipédia, le sens linguistique est la signification d'une expression (mot, syntagme, phrase, énoncé...), c'est-à-dire l'idée qui y est associée, dite aussi dénotation, avec ou sans connotation.

l'idée associée est avec/ou :

- la signification
- l'idée représentée par un mot / une phrase
- l'idée que veut véhiculer une personne quand il parle ou en écrivant une oeuvre littéraire / artistique

Dans les **dénotations**, il y a un lien entre le **signifié** c'est à dire l'objet en lui même comme un avocat, et le mot ce qui est écrit **signifiant** (Saussure). Et plus communément, dans un dictionnaire, un mot peut avoir différents sens. Les taxinomies, sont des dictionnaires pour les ordinateurs, qui sont comme de grands arbres. Il y a des relations entre les synonymes, les hypéronymes, et les hyponymes. Ces arbres sont codés à la main et sont difficiles à utiliser et à mettre à jour. Il y a Wordnet par exemple. Les linguistes, se mettent d'accord, pour faire des liens de rapprochement de sens entre un mot et un autre, comme thé et théière par exemple.
Le codage des taxinomies est compliqué, car la pensée humaine est faite en catégorie, car plus utiles pour raisonner, on a du mal à classer le sens, dans un espace vectoriel (vecteur one hot), dans un contexte discret. Ensuite, il est difficile de mettre à jour un mot, pour savoir où on doit placer le mot dans l'arbre. Il n'y a pas de distance graduée, c'est soit un synonyme, un hypéronymes ou hyponymes.
La similitude ici avec le vecteur one-hot, c'est que c'est quelque chose d'absolu, c'est à a dire qu'un mot par rapport à un autre est dans l'une des trois catégories. Enfin le dernier problème, c'est la dimension, on ne peut pas faire un vecteur one-hot aussi grand.

Comme nous l'avons déjà vu, les vecteur one-hot sont tous orthogonaux les uns avec les autres. Pourtant certains mots sont proches les uns des autres. Comment peut on déterminer la proximité ?

20.1.2 Type et Token

Différence entre Type et et Token:

La **Tokenisation** est par définition le découpage d'un texte en token, ce découpage très diffèrent en fonction des langues. En effet, ça dépend du type de la langue, en fonction de celle-ci, la tokenisation peut s'avérer compliquée, rien que pour les espaces par exemple , il a y les inversions les "qu'il" etc.

Un **type** c'est en quelque sorte, les différentes formes de mots, on les stocke alors dans un **dictionnaire**, et **tokens** ce sont les occurrences de ces formes se trouvant dans le dictionnaire. Le type est en quelque sorte une classe, et les tokens ce sont les différentes occurrences, il y a donc plus de tokens que de classe. Si un mot est répété beaucoup de fois, par exemple dans la phrase "bla bla bla", il y a 1 type, mais 3 tokens, le type est "bla" qui est stocké dans le dictionnaire et le token répété trois fois est "bla".
Pour les types: "Aimer", "aimerions", etc, ce sont tous des mots qui ont le **même type** (en français, 70 formes par verbes par exemple).

Qu'est ce qu'un **dictionnaire**, il y a deux sens. Au sens **informatique**, pour nous c'est juste une liste de mots, *token-to-indice*, c'est un dictionnaire, chaque forme apparaît une seule fois, par exemple "aimerions" apparaît dans notre liste car il est dans le corpus, mais on a aussi "aimer". De plus, une informations supplémentaire est apportée, chaque forme a une **fréquence** d'apparition, un **vecteur**, une **traduction**.
Enfin, sous la forme **linguistique**, la forme de **lemmes**, il n'y a pas "aimerions", il ne se trouve pas le dictionnaire, les lemmes sont les formes de base, il n'y a que le verbe à l'infinitif (singulier, genre), ce qui propre aux langues européennes. Un verbe français, donc sous la forme de lemme à l'infinitif, peut avoir avoir **70 formes flexionnelles**.

Quotidiennement lorsque nous parlons, nous utilisons en moyenne 20 000 tokens dans notre langue maternelle. Cette statistique dépend aussi des personnes et peut monter à 100 000, ça peut dépendre de la richesse de la langue, et des différences de vocabulaires entre chaque individus. Lorsqu'on connaît plus de 2000 **lemmes** pour une langue donnée, on peut considérer qu'on parle bien cette langue (sachant qu'un verbe conjugué compte pour un seul type, mais qu'il correspond à plein de *lemmes*). Cette statistique dépend dont des langues et de la manière de compter. Il y a des millions de mots pour chaque langue (en incluant les noms propres), donc c'est infaisable avec les vecteurs one-hot.

Pour les tokens, on compte les répétitions de chaque mot dans le texte. Dans la première partie on se rend compte qu'il y a beaucoup de nouveaux de mots, mais au fur et à mesure on en aura moins, car les mots vont se répéter.

On parle de richesse de vocabulaire grâce à la relation type/token, le ratio du nombre de type divisé par le nombre de tokens (ce ratio est toujours inférieur à 1). Dans un texte, plus le ratio est grand, plus l'auteur a un vocabulaire riche cependant ça n'a du sens que si les deux textes sont de même taille. Reprenons notre exemple, "bla bla bla" = 1 type 3 token , donc 1/3 (pas terrible)

Plus un texte grandit, moins on a de nouveaux mots. Il est alors difficile de comparer deux textes qui n'ont pas la même taille. Il faut prendre deux textes ou deux échantillons de même taille pour qu'il y ait un intérêt à comparer. On remarque que le ratio type/token est représenté par une courbe qui converge vers 1 en fonction de la taille du texte.

Un token diffère d'une langue à une autre. En anglais par exemple, don't, combien de mot on a ? Où est ce qu'on coupe ?

- Beaucoup de fois ce qu'on fait pour l'anglais on a des règles spéciales, on coupe avant l'apostrophe pour récupérer "do" et "not" de "don't"

- on français on coupe après, "vient-il" pour récupérer "vient" et "il", "l'humain"

On peut compter les ponctuations comme token (pour notre cas, on ne le fera pas).

Dénotation : C'est un objet sur lequel on pointe, l'objet qui est décrit
Connotation : C'est tout ce qui s'ajoute en plus, les idées que ça évoque pour nous (ou partagées, qui évoquent beaucoup de choses pour plein de monde)

Par exemple une théière, ce sont tous les objets qui remplissent ce critère, et cet avis n'est pas forcément partagé entre les différentes personnes, comme par exemple aimer boire du thé, dans quel contexte on aime le boire etc. On peut aussi voir théière avec ou sans manche (même chose pour une chaise sans pied ? Etc). Peut-être le thé est il-associé à un quelque chose de négatif pour nous, parce qu'on n'aime pas en boire.

On veut avoir les mots dans un espace métrique, des distances qui correspondent à une distance sémantique (le sens). Par exemple, bateau et navire n'ont pas la même orthographe mais ont plus ou moins le même sens. On aimerait que plus un mot soit proche de l'autre dans l'espace, plus les deux veulent dire la même chose. Ça peut permettre à un moteur de recherche de trouver des documents similaires.

20.1.3 Le contexte d'un mot

L'idée ici est de définir implicitement le sens d'un mot. On va regarder dans quel **contexte** apparaît un mot, on va alors pointer le sens, comme par exemple une chaise ici a 4 pieds.
Cet idée a été révolutionnaire: **Si on sait où on peut insérer un mot dans une phrase, c'est qu'on sait ce qu'est le mot/son sens, on n'a alors besoin pas de la connotation**. Si on est confronté à une phrase à trou, on se demande alors si on peut placer un mot à cet endroit. Pour cela, on doit alors connaître les premiers mots/ceux autour. C'est l'idée à la base du **plongement**.
Si on ne parle pas une langue, et si on nous donne une phrase à trou, on ne saurait pas quoi mettre dedans. Mais si on a la même phrase en français, on pourrait déduire, quel mot en français on pourrait introduire dans cette phrase en français.

Pour revenir à notre exemple, le mot théière, est ce que qu'on sait où mettre ce mot la dans telle phrase; si on sait le faire, c'est qu'on connaît la langue. Sinon c'est qu'on ne la connaît pas.

L'idée c'est que connaître un mot revient à savoir dans quel contexte il peut apparaître.

Si quelqu'un pose une question mais qu'on ne comprend pas ce qui est demandé c'est un peu différent, ce n'est pas ce qu'on cherche à faire ici. Poser des questions ici peut nous amener à certaines choses, comme des chatbots qui répondent à des questions car il ont vu la même réponse plusieurs fois répétée, comme "J'ai 40 ans", qui est un même. C'est le domaine de la "génération automatique de texte", qui vise à créer par exemple des chatbots.

On sait faire des phrases parfaites qui sont grammaticalement correctes mais qui n'ont pas de sens fondamentalement.

On a les **synonymes**, par exemple **vélo** et **bicyclette**. On constate qu'on peu de vrais synonymes car la plupart du temps, on utilise certains mots en fonction du niveau de langage, l'un est familier l'autre est standard, par exemple pour "pieu" et "lit".

Ensuite, on a les **homonyme** : avocat, c'est un hasard entre les deux, qui est historiques, car les gens ont pas fait attention, les deux ont pas gêne car sont utilisés de façon naturelle En on a les **polysème** : lit (l'objet ou verbe lire), bureau (l'objet ou l'espace de travail),

Pour les expressions **multi-mots** par exemple " pomme de terre", qu'aimerait on avoir dans la tokenisation ? Il ne faudrait pas que la signification dépende trop de *pomme* ou de *terre*.

En français et anglais on met des espaces entre tout ce qui a un sens, alors qu'en chinois ce n'est pas le cas, on parle d'espace entre les mophèmes (la plus petite unité qui a un sens en langage).

Il y a les **Entités nommées** (par exemple place de la concorde), mais aussi **terminologies** (les termes de mots, des suites de mots simples, par exemple en anglais "support vector machine"? c'est un terme, on aimerait avoir un unique embedding, il faudrait un processus itératif pour le reconnaître, car pris séparément, les trois tokens "support", "vector", et "machine" ne sont pas compositionnel. Savoir ce que veut dire chaque terme séparément non suffit pas à savoir ce que veut dire l'expression.

Toutes commes les terminologies, les **idiomes** par exemple "casser sa pipe", nécessitent un embedding particulier, car en connaissant "casser" et "pipe", on ne sait pas ce que veut dire cet idiome, on ne peut pas seulement se baser sur les unités séparées par un espace

Revenons à "pomme de terree", qui a 3 vecteur, on aura alors le token "pomme" qui vas être perturbé par pomme de terre.

Pour la phrase "Il couvre la pomme de terre", il y a y e ambiguïté du découpage, on sait pas si il couvre "la pomme" ou s'il couvre "la pomme de terre". On a pas de point de représentation pour les multi-mots

Le contexte d'un mot est aussi très important, comme par exemple avec le mot avocat, qui est ambigu, entre l'aliment, et le métier. La plupart du temps on ne se rend même pas compte de cette différence tellement c'est naturel pour nous. Exemple de d'utilisation du mot avocat:

- Moins de deux mois après avoir soumis à la profession d'*avocat* et aux juristes d'entreprise un chapitre de son projet de loi créant de façon expérimentale un statut d'*avocat* salarié en entreprise, le ministre de la justice a décidé d'enterrer purement et simplement le projet.
- Sur la requête du parquet général de la cour d'appel de Paris, les magistrats de cette juridiction ont infirmé l'inscription de M. Zadi en tant qu'*avocat*.

- Non, le noyau d'*avocat* ne se mange pas comme l'a laissé croire une vidéo qui a fait le buzz récemment, et comme l'ont repris de nombreux blogs et sites féminins. Aucun test ni recherche n'ont été entrepris sur la consommation de noyaux d'avocats par l'homme, et même pas par les animaux, donc la prudence s'impose.
- Étant un arbre fruitier subtropical, l'avocat ne fait pas du tout bon ménage avec le froid. Pensez à protéger l'arbre fruitier du gel en hiver.

Autre exemple : Dans Orange Mécanique, l'auteur utilise "Alex est sorti avec sa bande de drougs". L'usage d'un terme russe n'est pas un problème, car on comprend le sens avec le contexte et lorsqu'on le revoit on comprend.

Les linguistes ont travaillé sur les phrases ambigus, il y a l'ambiguïté lexicale:

- "La petite ferme la porte" Ce sont deux structure de la même phrase qui n'ont rien avoir, et qui nécessitent deux analyses différentes, ferme qui vient de la ferme ou bien du verbe fermer
  - "La petite" ferme la porte, ici ferme = verbe et porte = objet
  - "La petite ferme" la porte", ici ferme = bâtiment d'agriculture et porte = verbe

  A l'époque, une ferme était un bout de terrain cédé par le roi de façon **ferme**, mais ce sens c'est perdu avec le temps. On peut alors se poser la question si les deux mots sont homonymiques ou polysémiques ? On pourrait penser qu'ils sont homonymes, aujourd'hui on ne voit pas qu'il y a un lien entre les deux.
- "Sophie sent la rose" => Sophie a l'odeur de la rose ou sent une rose?

On remarque qu'on peut par élimination deviner le sens d'un mot, on a déjà une idée du domaine. En effet, en fonction du contexte dans lequel on voit le mot : dans un article scientifique, un article sur quelqu'un etc.

Pour les mots d'une autre langue (proche de la notre), on peut avoir une idée du sens, par exemple si ce sont des langues qui ont la même origine que la notre, comme les langues latines/européennes. En effet, juste avec le contexte, on peut au moins deviner le domaine, comme un article scientifique ou de philo.

20.1.4 Plongement - cours numéro 3 (à fusionner avec le reste)

Fun fact:
Ghoc: se lit commme "fish" si on pense a
gh comme dans laughing,
o comme dans women
c comme dans delicicious

Le plongement de mot ou plongement lexical (word embedding) est une méthode d'apprentissage d'une représentation de mots utilisée notamment en traitement automatique des langues mais un terme qui la représenterait mieux serait celui de la vectorisation de mots. Elle est basée sur l'hypothèse "de Harris" ou "distributional hypothesis" selon laquelle les mots apparaissant dans des contextes similaires ont des significations apparentées.

Le but de cette méthode est de représenter chaque mot d'un dictionnaire par un vecteur de nombres réels. La particularité de cette représentation est que les mots apparaissant dans des contextes similaires possèdes des vecteurs correspondants relativement proches. Par exemple, les mots "chien" et "chat" seront représentés par des vecteurs relativement peu distants dans l'espace vectoriels où ils sont définis.

Cette méthode permet notamment de prédire un mot dans un texte à trou uniquement avec quelques phrases grâce au contexte donné. Par exemple, les phrases

- Une bouteille de _____ est sur la table.
- Tout le monde _____ aime .
- Ne buvez pas trop de _____ avant de conduire.
- Nous faisons du _____ avec du maïs.

donnent un contexte qui définit uniquement un mot : tesgüino.

Cependant, deux questions se posent pour cette méthode.

20.1.5 Problème des différentes langues - cours numéro 3 (à mettre en lien avec le reste)

La première question à se poser est celle de l'efficacité de cette méthode dans certaines langues. En effet, différentes langues peuvent apporter différents problèmes.

Par exemple, l'allemand est composé de mots extrêmement longs car souvent composés d'autres mots. Ainsi, le mot *Donaudampfschifffahrtselektrizitätenhauptbetriebswerkbauunterbeamtengesellschaft* signifie 'Société de sous-traitance d'ingénierie des opérations principales d'électricité pour la navigation à vapeur sur le Danube'. Mais d'autres mots plus courants sont aussi dans ce cas. On pourrait citer le mot 'abus' qui se dit 'mal-usages' (Missbrauch) et 'poussettes' se dit 'chars d'enfants' ('Kinderwagen').

De plus, un mot n'est pas toujours une séquence de lettres, les mots composés avec des espaces posent par exemple encore plus de problèmes. Le mot 'pomme de terre' pourra être pris comme trois mots séparés dans le plongement de mots, changeant sa signification de s'il était pris en un mot. De plus, il y a des mots séparés qu'on ne veut pas séparer car n'ayant pas de signification l'un sans l'autre (Burkina Fasos). Il est à noter que le seul nom commun de la langue française dans ce cas est le mot "aujourd'hui" car "aujourd" et "hui" ne veulent rien dire.

Ainsi, le plongement dans ces cas n'est pas facile à faire et nécecssite un prétraitement des données pour savoir où séparer les mots.

D'autre part, certaines langues n'ont pas d'alphabets de 'mots' mais plutôt de caractères, comme le chinois ou le japonais. Ces caractères posent d'autres problèmes. Le chinois comme le japonais, par exemple, n'espace pas ses termes. Le japonais présente cependant une simplification en raison de la variété de son alphabet car en général, un changement de type de caractère indique un nouveau mot. Dans ces langues, bien qu'il y ai pour chaque mot des écritures et des phonétiques différentes, chaque caractère a une signification sémantique. Dès lors, on distinguer deux types d'écritures.

Les langues plus problématiques sont celles dites agglutinatives, comme le coréen, le japonais et la plupart des langues d'Asie Centrale. Le principe de ces langues est que tout un complexe verbal (les verbes modaux, auxiliaires, etc) rentrent dans le verbe souvent situé en fin de phrase. Ainsi, en japonais, l'expression "J'aurais aimé être embrassé" ne se dit qu'en un seul mot. Ceci entraîne beaucoup de variations pour des expressions très similaires et des mots dont les occurrences sont très rares (et apparaissent souvent une unique fois sans qu'on puisse en déterminer le contexte). On apprends alors juste des suites de caractères. Ce problème de rareté des objets nécessite un preprocessing très compliqué. Le coréen, en particulier, est une langue phonétique mais dans laquelle chaque symbole est englobé comme un seul caractère. Ainsi, pour effectuer un plongement, il est nécessaire de commencer par désemboiter les caractères.

Les langues isolantes sont des langues sans flexion, il n'y a pas de variante d'un terme, il n'y a pas de conjugaison et de déclinaison, comme le chinois par exemple, il y a d'autres langues comme le créole qui s'est crée récemment (et d'autres comme en Afrique, la plupart sont des langues sinétiques(qui viens de la chine) le tibétain etc).

Pas d'interaction entre les mots, les mots ne changent pas leur forme en fonction du contexte dans lequel ils apparaissent. Par exemple le verbe " fermer" devient "ferme" quand on le conjugue, c'est ce qu'on appelle la flexion. Il y a aussi quand le mot est sujet, complément d'objet etc. En français, on a les élisions, l'effacement d'une voyelle en fin de mot devant la voyelle commençant par le mot suivant "le arbre" qui devient "l'arbre". L'élision est une forme de contraction, les autres contractions sont par exemples "aujourd'hui" qui est la contraction de "au", "jour", "de" et "hui", comme " à le" qui devient "au". Les langues isolantes n'ont pas de **contractions**. De plus elles sont sans règle de **Sandhi**.

20.1.6 Problèmes des homographes - cours numéro 3 (à mettre en lien avec le reste)

L'autre problème à se poser est le cas des homographes, soient de deux mots ayant la même écriture mais dont la signification est différente. On peut distinguer deux types d'homographes différents.

Tout d'abord, les mots ambigus sont des mots s'écrivant de la même manière mais dont la signification est totalement différente (par exemple le mot 'avocat').

Les polysémies sont quant-à-elles des mots qui s'écrivent de la même manière et dont les significations, bien que différentes, se rejoignent (par exemple le mot 'bureau' qui peut signifier l'objet ou la salle, salle dans laquelle on trouve souvent l'objet). Les polysémies apparaissent souvent dans un même contexte. Les polysémies peuvent aussi indiquer un manque de nuance dans le vocabulaire d'une certaine langue. Le mot 'poisson' en français désigne ainsi aussi bien le poisson que l'on mange et celui qui vit dans la mer. De par la proximité de leur contexte, les polysémies sont moins gênantes que les mots ambigus.

Cependant, les polysémies peuvent mener à des phrases ambigues : dans la phrase 'je n'aime pas travailler au bureau', on ne sait pas si le locuteur n'aime pas travailler sur ou dans un bureau. Ce problème n'a souvent

pas lieu avec les mots ambigus, car le contexte suffit à deviner de quel sens du terme il s'agit, et le locuteur sait clairement de quel 'avocat' (par exemple) il parle, ce qui n'est pas toujours clair dans le cas des polysémies.

Les mots ambigus posent un problème puisqu'on risque de se retrouver dans un cas ou 'juge' se rapprochera de 'concombre' à cause du mot 'avocat', notamment avec les grandes dimensions. Lorsque l'on souhaite effectuer une PCA (ou une réduction dimensionnelle en général), le mot 'avocat' risque de se retrouver très loin des deux clusters auxquels il devrait être attribué.

20.1.7 Loi de Zipf (?à mettre en lien avec le reste? - si possible?)

C'est une loi assez problématique dans le NLP. Nous allons l'illustrer avec un exemple: dans le journal *Les Echos*, *Le monde* et L'oeuvre de J.P Sarte, le mot le plus utilisé est "de", cependant, ce n'est pas le cas pour *La disparition* de Perec. Ci-dessous est une illustration de la loi:
Ensuite allons "lissé l'histogramme":

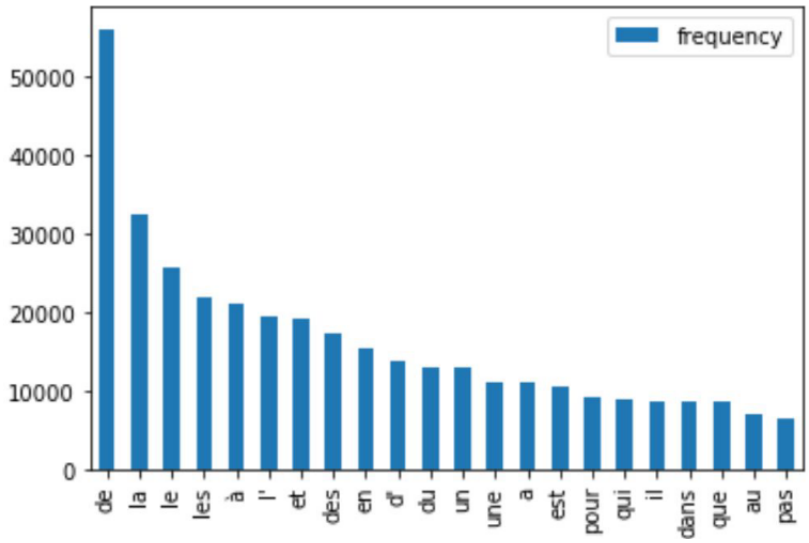
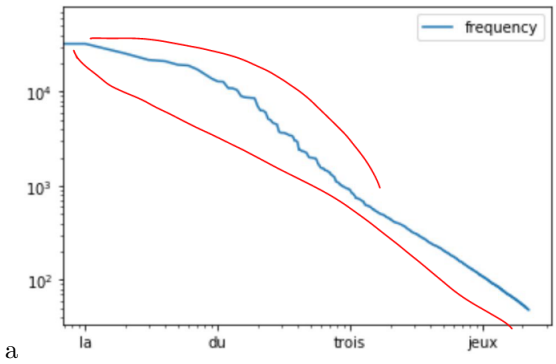


Figure 14: Distribution de mots dans un texte

Enfin, pour un résultat plus parlant, nous allons passer à une échelle logarithmique:



a

20.2 Plongements: intuitions mathématiques

20.2.1 Comment encoder l'idée qu'un mot est défini par son contexte

On souhaite entraîner ce qu'on appelle un word2vec, c'est-à-dire un plongement depuis l'espace du vocabulaire, l'espace des mots (*word*), c'est-à-dire depuis un espace à $V \approx 20000 - 200000$ dimensions, vers l'espace de plongement, qui a lui seulement $D \approx 50 - 200$ dimensions. L'entraînement se fait sur des tâches qui sont, formellement, des classifications, dans tous les cas. Mais ces classifications ne sont que des prétextes pour entraîner un bon plongement (*embedding*). Il y a deux familles d'algorithmes que nous allons développer dans les parties qui suivent, les **skip-gram** et les **Bag of word** (BOW).

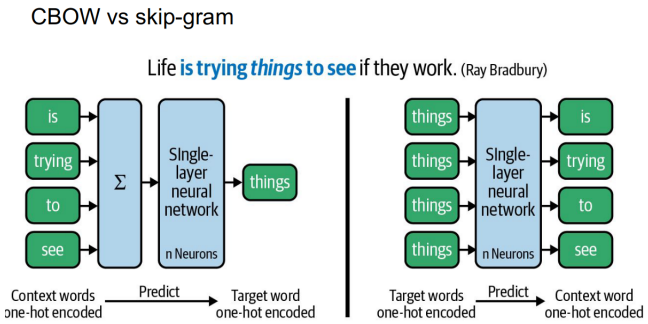


Figure 15: A gauche: CBOW, à droite: skip-gram

- le modèle CBOW (pour Continuous Bag-Of-Words): l'idée de ce modèle est de trouver un mot à partir du contexte.
- le modèle skip-gram: c'est l'inverse de l'idée précédente, d'un mot central, nous essayons de déduire le contexte.

Illustrons cela avec un exemple: voir figure 15.

20.2.2 Skip-gram

Idée générale: **étant donné un mot, on veut prédire un contexte qui va bien avec** (en fait, d'autres mots qui vont bien avec).

Étant donné un mot central (dans l'exemple, *things*, ce sera le x_n, le mot-input), on veut entraîner un modèle qui puisse prédire des "mots cibles", c'est-à-dire les mots qui sont autour de ce mot, dans la phrase (dans l'exemple, *is*, *trying*, *to*, *see* seront des y_n, des mots-cibles). On définit la notion de "autour de" par la proximité dans la phrase, en termes de mots espacés par un espace, donc avec une fenêtre de taille prédéfinie (dans l'exemple, elle est de taille 2: 2 mots avant le mot-input, 2 après).

Intuitivement, ça devrait marcher: si on nous donne par exemple une phrase en allemand qui contient le mot Kern, on sait qu'on est dans un contexte où on parle de math, et on va donc chercher des mots proche des maths. Autre exemple, si on prend une fenêtre de 2 (2 mots avant et après) sur la phrase "Pensez à *protéger l'**arbre** fruitier du* gel en hiver". Ici on aura une grande probabilité de trouver *fruitier* dans le contexte de *arbre*. En revanche le terme *protéger* est moins associé au terme *arbre*, il sera donc classé moins haut comme étant du même contexte (probabilité d'apparition dans la fenêtre plus basse).

Ici, les inputs X sont donc des mots situés au centre d'une fenêtre de mots, dans une phrase, et les outputs à prédire sont les mots Y qui apparaissent autour de ce mot central. On fait cela pour tous les mots. C'est donc de la **classification supervisée**, en quelque sorte, même si **il y a plusieurs "bonnes réponses" par mot-input, et qu'il y a V classes**, $V \approx 20000$ par exemple étant la taille du vocabulaire.

On va détailler plus bas la construction de l'ensemble d'apprentissage, mais on peut déjà noter que si la fenêtre est de taille m, c'est-à-dire qu'il y a $2m+1$ mots dans la fenêtre (m à gauche, $+1$ le mot central, $+m$ à droite du mot central), on a $2m$ paires de mots, donc $2m$ exemples dans notre *training set*. En fait, dans une phrase, les m premiers mots et m derniers mots n'ont pas exactement $2m$ voisins, donc ça fait un peu moins, mais bon, l'idée est là: chaque mot du corpus va permettre de produire de nombreux exemples.

Ce modèle va être explicité en détails dans une section plus bas.

20.2.3 Bag Of Words

Idée générale: **étant donné un contexte** (une série de mots), on veut **prédire un mot qui va bien avec**.

On peut considérer que BOW est l'inverse de skip-gram puisse ce que cette fois-ci, on utilise le contexte pour trouver un mot central. Pour reprendre l'exemple précédent, on donne "protéger l'??? fruitier du" et le modèle doit deviner "arbre". Dans le premier exemple, on a "is trying ?? to see" et on doit prire "things".

Intuitivement, ça devrait marcher, au moins si on accepte la définition du sens des mots données par Wittgenstein deux, c'est-à-dire que "connaître une langue, c'est savoir compléter des phrases à trou"[9] (on paraphrase un peu par pédagogie).

[9]La citation exacte est « La signification d'un mot est son usage dans le langage. » – Wittgenstein, Recherches philosophiques, 1953

Ici, les **inputs X sont donc des séries de $2m$ mots** (des contextes), et les **outputs Y sont des mots-cibles** Il y aura donc N exemples d'apprentissage pour un corpus de N mots. Certains exemples, concernant les mots cibles en début et fin de phrase, auront moins de contexte (car pas de voisins à gauche, à droire, respectivement). Ici encore, c'est une **classification supervisée**, les inputs sont un peu plus riches, et **il y a V classes**. Il peut arriver qu'une phrase à trou ait plusieurs bonnes réponses, mais c'est plus rare.. ça n'est pas un point très important, car de toute façon **le but est de produire des probabilités non nulles pour tous les mots qui pourraient crédiblement se mettre dans le trou** (et souvent, il y en a plus d'un, même si la phrase a trou n'est apparue qu'une fois dans le corpus).

20.2.4 Intérêt des deux approches

Bizarrement skip-gram produit de meilleurs résultats que BOW, mais il mets plus de temps pour obtenir des résultats cohérents. C'est dur d'intuiter pourquoi, une piste est peut-être dans le fait que skip-gram doit prédire plusieurs mots, alors que BOW en prédit un seul.

Ce qui est pratique avec ces deux modèles, c'est qu'en minimisant une fonction coût (L comme *Loss*, Perte en anglais), on obtient **une représentation où les mots sémantiquement proches sont proches les uns des autres**. Autrement dit, on obtient une distance dans l'espace sémantique (l'espace du sens des mots... le sens, entendu au sens de "je sais compléter une phrase à trou = j'ai compris le sens du mot").

Un peu d'histoire: **il y a un avant et un après Word2Vec** (2013), Mikolov. Tout marche mieux avec l'implémentation type word2vec (via skip-gram ou BOW), donc **quasiment plus personne n'utilise les anciennes méthodes**.

En général pour l'apprentissage, on a besoin de besoin de données annotées, c'est-à-dire avec des labels disponibles. Par exemple une image avec des étiquettes (tags) qui décrivent ce qu'on y voit. C'est l'ingrédient indispensable à l'apprentissage supervisé. Mais **ces deux techniques n'ont pas besoin de labels: on les fabrique à la volée !** Du coup, toutes les données d'internet sont devenues une véritable mine d'or pour l'entraînement des ces modèles. On peut donc maintenant facilement analyser des commentaires, des tweets, détecter des spams, avoir un avis général d'un produit (positif ou négatif)...

Grâce à word2vec (skip-gram ou BOW), on obtient une vectorisation des mots, on peut maintenant appliquer toute la puissance des maths à nos mots.

20.3 Skip-gram en détails: comment entraîner un word2vec "fait maison"

20.3.1 Fonction coût

La première approche qu'on a vu est la suivante (skip-gram):

$$J^1(\theta) = \prod_{t=1}^{T} \prod_{-m \leq j \leq m; j \neq 0} P(w_{t+j}|w_t; \theta),$$

avec T les mots du texte, $-m < j < m$ la fenêtre, w_t le mot central et θ les paramètres. Ici, m, qui détermine la taille de la fenêtre, est un hyperparamètre[10]. Le terme $\vec{w}_t$ correspond à un identifiant unique (vecteur one-hot par exemple) du mot numéro t dans le texte. Les indices t ou $j + t$ dénotent l'indice du mot dans la phrase (la position du mot dans le corpus de texte d'apprentissage), donc t va de 1 à N, le nombre de mots dans le corpus. Pour le moment, on ne précise pas de modèle mathématique pour la probabilité jointe $P(a|b, \theta)$. C'est évidemment une question cruciale.

On peut passer l'équation sous forme logarithmique pour faciliter la dérivée:

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{-m \leq j \leq m; j \neq 0} \log(P(w_{t+j}|w_t; \theta))$$

On a ajouté le signe $-$ pour faire une minimisation, plutôt que de maximiser (c'est une habitude de matheux), et on a ajouté $\frac{1}{T}$ pour faire la moyenne par mot, ce qui permet de comparer des textes de tailles différentes. Ici on ne prends pas en compte la valeur précise de la distance ($|j|$) entre les mots dans la phrase, dès qu'ils sont à distance de moins que $|j| \leq m$, on les considère voisins, et c'est tout.

[10] On peut noter window_size au lieu de m si la lettre m est déja prise.. on essaiera d'éviter ce genre de collisions.

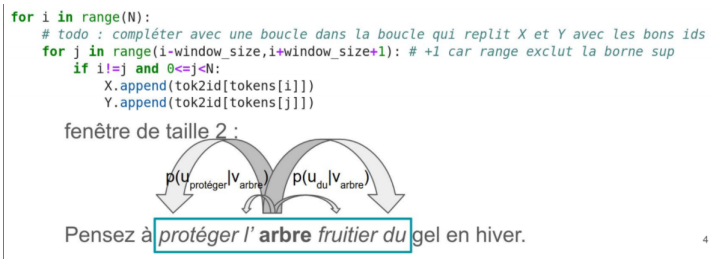


Figure 16: Code parcourant le texte pour créer les vecteurs X et Y

X	Y
297	1001
297	547
297	3
297	105
42	...

Figure 17: Exemple de vecteurs X et Y obtenus

20.3.2 Training set

Lorsqu'on fera l'apprentissage, comme on l'a vu dans sec. 20.2.2 et dans sec. 20.3.1, on va parcourir les mots du corpus, et utiliser à chaque fois un mot central comme entrée (mot-input x) et les mots autour comme mot-cibles (y). Si **un mot** à 4 voisins (2 avant, 2 après), cela **fera 4 paires de** (x, y)**, donc 4 exemples** ou point de donnée pour l'ensemble d'entraînement, même si on aura 4 fois le même x: hé oui, ça ne nous pose pas de problème d'affirmer qu'une même entrée à 4 bonnes réponses associées qui sont alternativement "correctes".

Notes plus détaillées, par et pour les étudiants: Voir la fig. 16 et fig. 17

Pour faire la boucle demandée, il faut comprendre ce qu'est la fenêtre. Pour chaque mot d'indice i, on va regarder les mots avant, depuis i-window_size, et après, jusqu'à i+window_size. Sur le schéma, arbre est le mot central. Ici si window_size = 2, on va regarder les deux mots avant le mot central et les deux mots après le mot central. Les deux mots avant sont "protéger" et "l'" et les deux mots après sont "fruitier" et "du".
Dans le tableau token on a les différents tokens. Et dans tok2id on a récupère l'indice du mot correspondant dans les indices du dictionnaire. En effet, car on stocke les types dans un dictionnaire, et nous on a besoin des indices par rapport à ce dictionnaire. Donc en résumé, tokens, ce sont tous les différents apparations des mots (au moins une fois), et dans le dictionnaire, c'est seulement les types, c'est à dire des mots différents.
Ce qui est subtil ici, c'est que les vecteurs X et Y sont de la même taille, dans les deux tableaux, on fera correspondre l'objet d'indice i qui est le mot central dans le dictionnaire, dans X[i] au mot d'indice Y[i] c'est à dire un mot qui est dans la fenêtre.
*Et donc X[i] va apparaître autant de fois qu'il a de voisins, c'est-à-dire apparaitre 2*window_size fois (attentions pour les mots qui sont au début et à la fin qui ont moins de voisins).*

20.3.3 Intuition sur le modèle choisi pour $P(w_{t+j}|w_t;\theta)$

C'est bien joli de dire qu'on va minimiser une fonction de perte, mais encore faudrait il avoir une expression mathématique pour notre expression $P(w_{t+j}|w_t;\theta)$. Dans ce paragraphe on reste encore relativement abstrait, mais cela permet de donner l'intuition de pourquoi c'est crédible que les choses fonctionnent.

On peut oublier le détail des indices t, j dans $P(w_{t+j}|w_t;\theta)$ et regarder l'expression générale $P(\vec{y}|\vec{x};\theta)$, où $\vec{x}$ est le mot-input (vu comme du "contexte") et $\vec{y}$ est le mot-cible à prédire. Tous les deux sont des vecteurs one-hot de dimension V. Par ailleurs, **on introduit** $\nu_c = WE^T\vec{x}$**, le contexte associé au mot numéro** c **du vocabulaire**, $\vec{x}_c$. Le vecteur $\vec{\nu}_c$ est lui aussi de dimension V, mais n'est pas one-hot, il a des valeurs non nulles dans toutes ses composantes. Pour le moment, W et E sont de mystérieuses matrices, qui ont les bonnes dimensions, tout comme il faut. Dans les slides, on a noté ν_c le contexte et $\vec{u}_o$ le mot-cible, qu'on notera plutôt $\vec{y}$ ici.

On se donne le modèle suivant, pour la probabilité de rencontrer le mot numéro o dans le contexte identifié

par l'indice c:

$$P(o|c,\theta) = P(\vec{y}_o|\vec{x}_c,\theta) = \sigma(\vec{y}_o^T \cdot \vec{\nu}_c) = \sigma(\vec{y}_o^T \cdot WE^T\vec{x}_c)$$
$$= \frac{e^{\vec{y}_o^T \cdot \vec{\nu}_c}}{\sum_{w=1}^{V} e^{\vec{y}_w^T \cdot \vec{\nu}_c}}$$

Ici o, w sont des indices identifiants les mots parmi le vocabulaire, V. V est par exemple de taille 20.000. L'indice c est relatif à un contexte, **ERRATUM**: mais concrètement, c'est en fait l'indice du mot-input $\vec{x}_c$.

On va apprendre (au sens Machine Learning, on va "fitter") les paramètres contenus dans les matrices W, E qui caractérisent comment ν_c dépend du mot-input $\vec{x}_c$.

20.3.4 Détail du modèle utilisé dans skip-gram

Le *modèle* $f_\theta(x_n)$ consiste en un vecteur de V valeurs, qui sont **interprétées comme des probabilités** que le mot x_n soit voisin (dans la phrase) des différents mots du vocabulaire (qui est de taille V). La valeur numéro v du vecteur $f_\theta(x_n)$ correspond à la probabilité que le mot x_n soit proche[11] du mot numéro v de notre vocabulaire ($v = 1, \ldots, V$). Parmi ces V valeurs, qui somment à 1 (car elles correspondent à une probabilité bien normalisée), la valeur la plus haute sera prise comme étant le mot prédit: $\hat{y}_n = \text{argmax}(f_\theta(x_n))$, mais ce n'est pas cette prédiction qui est intéressante, en effet ce $\hat{y}_n$ est juste un indice d'un mot dans le vocabulaire. La donnée de la distribution complète des probas $f_\theta(x_n)$ est beaucoup plus riche d'informations que son argmax.

On introduit d'une part un **plongement** (*word embedding*) E, qui est une matrice de taille (V, D), où V est la taille du vocabulaire (de l'ordre de 20 000 ou 200 000) et D est la dimension choisie pour l'embedding, par exemple 50 ou 100, voire 200. Lorsqu'on multiplie **un mot, ou plutôt son vecteur one-hot correspondant** $\vec{x}_n$, par l'embedding E, on obtient la **représentation** de ce mot par le plongement, $\vec{r}_n$, c'est-à-dire un vecteur de D composantes, qu'on espère être une bonne représentation du mot: $\vec{r}_n^T = x_n E$, qui sera un vecteur ligne (ou bien sa transposée $\vec{r}_n = E^T x_n$ si on préfère avoir un vecteur colonne)

D'autre part, il y a une matrice W de dimensions (V, D) qui permet de **faire l'opération inverse**: partant d'une représentation comprimée d'un mot, $\vec{r}_n$, on retrouve une représentation brute, c'est-à-dire quelque chose qui ressemblerait à un indice dans le vocabulaire: $\vec{w}_n = W\vec{r}_n$, où $\vec{w}_n$ est un vecteur de taille V, avec des valeurs plus ou moins élevées. En principe on espère que l'indice où $\vec{w}$ est maximal correspondra à l'indice d'un mot similaire, $\vec{y}_n$, qui était le mot cible.

Comme indiqué plus bas, la sortie du réseau est choisie comme étant, moralement, $y \approx W.E^T.x_n$, c'est-à-dire qu'on compresse et décompresse le mot x_n, en passant par l'espace de plongement, dont la dimension D est très petite par rapport à V.

Ici, on peut faire une **comparaison avec la PCA**: en PCA, la matrice P, de dimensions (D, D'), permettait de comprimer les données naturelles vers un espace abstrait de dimension D'. La matrice P^T permettait ensuite de "décompresser" les données, c'est-à-dire de revenir à l'espace initial, de dimension D. Ici, on passe de la dimension V à la dimension D, E **joue un rôle semblable à celui de** P**, et** W **joue un rôle similaire à celui joué par** P^T. Cependant, la différence est que la PCA cherche à minimiser l'erreur de reconstruction entre une donnée x_n et sa version reconstruite $\simeq x_n PP^T$ (sans rentrer dans le détail), au lieu de chercher à avoir une proximité entre un mot x_n et ses voisins y_n. L'autre différence est bien entendu que avec la PCA, il y a une seule matrice, P, qui sert à compresser et décompresser (avec la transposée), alors que pour les plongements, W et E sont deux matrices distinctes.

On rappelle la définition du softmax: *(TODO: mettre p-e le softmax en annexe plutot que ici?)*

$$\sigma(\vec{a}) \hat{=} \text{softmax}(\vec{a}) \tag{135}$$

La k-ème composante du vecteur $\sigma(\vec{a})$ s'écrit:

$$\sigma(\vec{a})_k = \frac{\exp(a_k)}{\sum_\ell \exp(a_\ell)} \tag{136}$$

Le softmax est noté $\sigma(.)$ dans cette sous-section ci, mais ce n'est pas général, en fait en ML on note souvent σ la fonction non linéaire (fonction d'activation) qui est appliquée après la multiplication par une matrice de poids, mais cette fonction d'activation peut prendre toutes sortes de formes (ReLu, softmax, sigmoïde, ...). Le softmax permet de passer des nombres réels a_k, qui sont des valeurs arbitrairement grandes, à une suite de nombres qui s'interprètent comme des probabilités (car la somme des éléments du vecteur fait alors 1, et chaque élément est positif). En effet, au numérateur, avec $\exp(a_k)$, on a quelque chose de positif (car

[11] proche dans les phrases du corpus, c'est-à-dire en terme de contexte, donc on espère, **sémantiquement proches**.

exp() est positive). Au dénominateur, avec la somme des exponentielles, on a pile ce qu'il faut pour que le vecteur $\sigma(\vec{a})$ s'interprète comme une proabilité. En effet, si on fait la somme de ses composantes, on trouve: $\sum_k \sigma(\vec{a})_k = \frac{\sum_k \exp(a_k)}{\sum_\ell \exp(a_\ell)} = 1$. Par ailleurs il est clair que chaque composante est plus petite que 1, vu le dénominateur (qui contient le numérateur plus d'autres termes positifs). Ce qui est bien avec le softmax, c'est que si un des a_k est grand devant les autres, avec l'exponentielle, cette différence va devenir immense et on aura un max bien défini, voire assez vite un vecteur $\sigma(\vec{a})$ qui ressemblera a un one-hot (ce qu'on aurait directement si on utilisant un max dur (un hard-max? personne ne dit ça), $\sigma(\vec{a}) = \{1 \text{ si } k = \text{argmax}_\ell(\vec{a}_\ell), 0 \text{ sinon}\}$. Mais dans le cas du softmax, toutes les sorties sont non-nulles, ce qui est bien pour faire de la descente de gradient.

Le **modèle** $f_\theta(x_n)$ **s'écrit** comme le softmax de $W.E^T.x_n$:

$$f_\theta(x_n) = \sigma(W.E^T.x_n) \tag{137}$$

Où x_n **est un vecteur one-hot** (de taille V) qui représente le **mot-input** numéro n dans nos exemples.
De même, y_n **est un vecteur one-hot** (de taille V) qui représente le **mot-cible** numéro n dans nos exemples.

20.3.5 Détail de la fonction coût minimisée dans skip-gram

On définit **l'entropie croisée** (*cross-entropy*) entre deux variables aléatoires p, q ainsi:

$$H(p,q) = -\sum_{\omega\in\Omega} p(\omega)\log(q(\omega)) \tag{138}$$

Intuitivement, c'est une fonction qui caractérise la proximité entre deux distributions de probabilités, en général une vraie distribution p et son estimation approximative, q. Si elles sont identiques, l'entropie croisée est minimale et égale à l'entropie $H(p) = -\sum_{\omega\in\Omega} p(\omega)\log(p(\omega))$. Plus elles sont différentes, plus l'entropie croisée sera grande (voir aussi: divergence de Kullback–Leibler).

La fonction coût à minimiser est alors celle-ci (*Loss* en anglais, d'où le L):

$$L(\theta, X, Y) = H(y_n^{true}, f_\theta(x_n)) \tag{139}$$

$$= H(y_n^{true}, \sigma(W.E^T.x_n)) \tag{140}$$

Les paramètres à optimiser, qui sont les composantes des matrices W, E[12], seront trouvés par descente de gradient:

$$\vec{\theta} \leftarrow \vec{\theta} - \eta\vec{\nabla}_{\vec{\theta}} L(\theta, X, Y) \tag{141}$$

Plus concrètement :

$$W \leftarrow W - \eta\frac{dL}{dW} \tag{142}$$

$$E \leftarrow E - \eta\frac{dL}{dE} \tag{143}$$

où on a noté, pour une fois $\nabla_E L = \frac{dL}{dE}$, le gradient de la fonction coût (la *Loss*) par rapport aux coefficients de la matrice E, par exemple.

20.3.6 Descente de gradient pour une entropie croisée

Le calcul du gradient de la cross-entropie d'une fonction du genre $H(y_{GroundTruth}, \sigma(W.x))$ est fait en exercice (TODO) exo. **??**. On trouve toujours quelque chose de la forme:

$$\frac{\partial}{\partial w_k} H(y^{true}, \sigma(\vec{w}\cdot\vec{x})) = \frac{1}{N}\sum_n^N \vec{x}\left(\sigma(\vec{w}\cdot\vec{x}) - \delta(k, k^{true})\right) \tag{144}$$

Où $\delta(a,b)$ est le symbole de Kroenecker, et vaut 1 lorsque l'indice k correspond à l'indice vrai (k^{true} ou y^{true}). Le vecteur one-hot dont les composantes sont égales à $\delta(k, k^{true})$ est exactement le vecteur y_n (de taille V). On ne détaille pas le calcul ici, mais on peut prouver que les dérivées (gradients) sont celles ci:

$$\nabla_W L = \frac{dL}{dW} = \frac{1}{N}\sum_n \left(\sigma(W.E^T.\vec{x}_n) - y_n\right)(E^T x_n) \tag{145}$$

$$\nabla_E L = \frac{dL}{dE} = \frac{1}{N}\sum_n x_n^T \left(\sigma(W.E^T.\vec{x}_n) - y_n\right)^T W \tag{146}$$

[12] ça fait quand même $2VD$ paramètres ! Soit 2 fois 200 000 fois 100 par exemple, c'est-à-dire 40 000 000 de paramètres !

On peut vérifier la dimensionalité des équations. W et E sont de dimension (V, D), donc les gradients sont aussi attendus comme étant de même taille.
Pour l'équation (145), $(\sigma(..) - y_n)$ est de taille $(V, 1)$. Le terme $E^T x_n$ de dimensions $(D.V).(V, 1) \to (D, 1)$ (pour un n fixé, et si on note X l'ensemble des exemples avec une dimension (V, N), ça fait $(D.V).(V, N) \to (D, N)$). La somme et produit entre $(\sigma(WE^T X) - Y)$ de taille (V, N) et $E^T X$ de dimensions (D, N), correctement réalisée, fait donc un terme de droite de dimension (V, D), comme attendu.
Pour l'équation (146), il faut raisonner de même. Le terme $(\sigma(WE^T X) - Y)^T$ est de dimensions (N, V), et $X^T(\sigma(WE^T X) - Y)^T W$ est $(V, N).(N, V).(V, D) \to (V, D)$. En pratique, X est un filtre qui sélectionne des mots (numéro v) du vocabulaire ($v \in [1, \ldots, V]$), N fois, pour faire les mises à jour sur E.

Remarque d'implémentation. Les équations ci dessus sont implémentées dans un cadre ou en fait, on peut faire la multplication matricielle $E^T.X$ au départ, en sélectionnant simplement des lignes de E qui sont concernées (repérés par les indices v où les x_n ont leur "hot", leur valeur non-nulle), potentiellement avec répétition. Ce produit $E^T.X$ est de dimension (D, N) et peut être appelé "word-vec", en python, ce sera une référence vers les lignes correspondantes de E, qui contient vraiment des paramètres à apprendre. Lorsqu'on met à jour cet élément, au lien d'appliquer l'équation (146)), on fera simplement
"word-vec"$\leftarrow$"word-vec" $-\eta W^T \left(\sigma(W.E^T.X) - Y\right)$ (et alors les deux cotés de l'équation sont de dimension D, N, et tout va bien).

20.3.7 Mini-Batches

Tout ce qui est présenté ci dessus suppose qu'on fait une mise à jour *full batch*, c'est-à-dire qu'on met à jour en considérant toutes les paires (X, Y) = (mot-input, mot-cible) d'un coup (ce qui est le choix "normal", "habituel", par défaut). Ici, vu le nombre de paures d'exemples, ce n'est pas raisonnable, rien que pour une question d'espace dans la mémoire vive. On applique donc **la stratégie du mini-batch**: on découpe l'ensemble d'apprentissage (les N paires de $(\vec{x}_n, \vec{y}_n)$) en *batches*, c'est-à-dire en paquets de B exemples. Pour peu que N soit multiple de B, on a donc N/B batches de B exemples chacun. Les sommes présentées plus haut, qui sont sur $n = 1, \ldots, N$ sont alors remplacées par de plus petites sommes, sur $n = b\frac{N}{B}, \ldots, (b+1)\frac{N}{B}$, et on parcourt les *batches* en faisant (en amont) un boucle sur $b = 1, \ldots, B$.

20.3.8 Retour aux intuitions

Comme on l'avait dit en sec. 20.3.3, le plongement cherche à maximiser la proximité entre le mot $\vec{y}_o$ et le contexte $\vec{\nu}_c = WE^T\vec{x}_c$. Cette proximité est caractérisée par le produit scalaire $y_o^T \cdot \nu_c$.

On peut voir que c'est ça qu'on a maximisé en reconsidérant l'entropie croisée:

$$L(\theta, X, Y) = H(y_n^{true}, f_\theta(x_n)) = H(\vec{y}_n^{true}, \vec{\sigma}(W.E^T.x_n)) \tag{147}$$

$$= -\frac{1}{N}\sum_n \vec{y}_n^{true} \cdot \log(\vec{\sigma}(W.E^T.x_n)) \tag{148}$$

$$= -\frac{1}{N}\sum_n^N \sum_v^V (y_n^{true})_v . \log\left(\frac{\exp(W.E^T.x_n)_v}{\sum_\ell^V \exp(W.E^T.x_n)_\ell}\right) \tag{149}$$

Où on a insisté sur le produit scalaire $\vec{y}_n^{true} \cdot \vec{\sigma}(..)$, puis on l'a explicité comme une somme sur ses composantes, indexées ici par la lettre v. On poursuit le calcul, pour un point de donnée numéro n fixé:

$$\sum_v^V (y_n^{true})_v . \log\left(\frac{\exp(W.E^T.x_n)_v}{\sum_\ell^V \exp(W.E^T.x_n)_\ell}\right) = \sum_v^V (y_n^{true})_v . \exp(W.E^T.x_n)_v - \log\left(\sum_\ell^V \exp(W.E^T.x_n)_\ell\right) \tag{150}$$

$$= \sum_v^V (y_n^{true})_v . (W.E^T.x_n)_v - \log(\text{constante}) \tag{151}$$

$$= \vec{y}_n^{true} \cdot (W.E^T.\vec{x}_n) - \log(\text{constante}) \tag{152}$$

$$= \vec{y}_n^{true} WE^T\vec{x}_c - \log(\text{constante}) \tag{153}$$

ce qui ressemble bigrement au modèle annoncé plus haut (à un softmax près.. détail à régler). À l'époque on l'avait noté $\sigma(\vec{y}_o \cdot \vec{\nu}_c)$

$$P(y_o|x_x, \theta) = \sigma(\vec{y}_o \cdot \vec{\nu}_c) \simeq \vec{y}_o \cdot \vec{\nu}_c = \vec{y}_o WE^T\vec{x}_c \tag{154}$$

On a bien maximisé ce qu'on avait annoncé.

Par ailleurs, on remarque $\vec{y}_o W E^T \vec{x}_c = (\vec{y}_o W)_{(D)} \cdot (E^T \vec{x}_c)_{(D)}$: c'est aussi le produit scalaire de deux vecteurs de dimension D, qui sont les vecteurs dans l'espace de plongement (dans l'*embedding*, on pourrait dire). **Ce produit scalaire correspond à une mesure de la proximité des vecteurs dans cet espace**. En fait, si on normalise tous les mots (le vecteur de dimension D qui correspond au mot), ce produit scalaire peut être vu comme le cosinus de l'angle entre les deux vecteurs.

On affirme et on utilise plus bas le fait que **la proximité des vecteurs dans cet espace de dimension D, correspond à une proximité sémantique (proximité du sens)**.

20.3.9 Structure et flux de l'information dans le Word2Vec - cours numéro 3 (à fusionner avec le reste)

TODO: cette section est elle vraiment utile ?

1. Vocabulaire
    - on décide de remplacer les mots rares par _unknown_, cela peut être fait si le texte est très long par exemple.
    - tok2id, un token (par exemple avocat) à qui on associe un id (par exemple 177). Nous pouvons imaginer cela comme un dictionnaire.
    - Même principe que précédemment, en inversant ce qu'est la clé et ce qu'est la valeur.
2. Initialisation
    - Choix de la taille de plongement, par exemple m = 50
3. Créer les couples (X,Y)

Définissons 2 matrices :

- Word embedding
- Dense

Il existe une étape délicate : Multiplier la matrice word embedding avec la matrice dense pour avoir une matrice de dimension (v,m). Après application d'un Softmax: on a une série de probabilité, utilisé pour la comparaison avec la réalité (ce que l'on souhaite). Notre entrée est de dimension m et notre sortie également.

Nous pouvons regrouper les opérations nécessaires en 2 catégories:

1. opération forward
    - ind_to_word_vec
    - linear_dense
    - Softmax
2. opération backward
    - Softmax_backward
    - Dense_backward

Tout cela est très lourd à calculer! Il existe des solutions:

- réduire le vocabulaire: mais on perds le traitement des mots rares (on perds un peu le point clé du word embedding).
- plus malin: L'échantillons négatifs, l'idée: utiliser les contextes connus, on met seulement à jour certains mots contextuels.

20.3.10 Différence entre window_size et batch_size - cours numéro 3 (à fusionner avec le reste)

Cette section sert à lever un malentendu lié à la notation m pour le batch size et le **Window_size**. Elle ne sera pas nécessaire l'an prochain.

window_size est pour tout entier i, la fenêtre d'association entre les mots d'indice i entre les vecteurs Y et X du même indice. C'est à dire la fenêtre de correspondance entre les mots d'indice Y[i] et X[i], en effet on rappel que X[i] et Y[i] sont des entiers qui renvoient aux types dans le dictionnaire.

Donc à l'aide de window_size on obtiens des vecteurs X et Y qui sont très grands, plusieurs fois le corpus. Il est alors difficile de mettre à jour, on va s'aider d'un hhyper-paramètres, le batch_size. L'objectif est donc de choisir le nombre d'éléments qui seront utilisés en même temps pour mettre à jour le gradient. Donc **batch_size** définis la taille des sous parties des vecteurs X et Y lesquels on travaille , et représente aussi le nombre d'échantillons à chaque mise à jour du gradient. Si **batch_size** $= 1$, c'est difficile de mettre à jour car on ferait la mise à jour un peu par hasard, on peut aussi faire tout à la fois, ce qui est coûteux mais ne sert pas forcément à grand chose. On peut alors bout qu'une petite partie, comme par exemple $batch\_size = 500$, on se concentre alors sur cette partie et on "oublie" le reste, et on met à jour la distribution. On aimerait mettre à jour cette distribution pour qu'elle arrive à prévoir les images qui sont alors dans le vecteur Y, en se basant sur le vecteur X qui sont les antécédents.

Ce sont deux hyper-paramètres différents qu'on doit définir. Ils sont difficiles à bien gérer, ça dépend de la taille du vocabulaire, de la taille du texte etc, une grande partie du travaille sera alors de trouver les meilleures valeurs pour maximiser les résultats des prédictions.

20.4 Application des plongements vectoriels

Dans la section précédente, on s'intéressait à la distribution de probabilité qu'un mot se trouve dans le voisinage du mot central, dans cette section, on va s'intéresser ce qui "oblige" cet espace vectoriel à placer certains mots à côté du mot central, c'est aussi ce qu'on cherche à minimiser. Les mots dans le voisinage sont alors sémantiquement proches. On s'intéressera aussi aux différentes applications du TAL et ce qu'on peut faire avec.

[Reliquat de séances précédentes:]
Parsing Syntaxique = domaine de prédilection de Kim Gerdes.
Oxymore: est ce que les machines peuvent reconnaître ce genre de figures de style ? C'est délicat, évidemment. beaucoup de chercheurs travaillent sur l'analyse de sentiment: savoir si un discours émet un avis positif ou négatif sur tel sujet. Cependant, même si beaucoup de chercheurs travaillent dessus, la plus part du temps, on a du mal à classifier les sentiments et l'ironie car ils dépendent beaucoup du contexte.

21 [Hors programme] Réduction dimensionelle: le lien entre plongements et PCA

voir la section 20.3.4.

Essentiellement: lors du plongement vectoriel fait pour le traitement des langues, on a fait une énorme réduction dimensionnelle, comme avec la PCA. On est passé de $V \approx 200000$ à $D \approx 100$! Contrairement à la PCA, on a 2 matrices différentes, W, E (*word embedding*) et non pas P, P^T (en PCA).

Mais l'idée reste qu'on minimise une sorte d'erreur de reconstruction:

- PCA: on minimise l'erreur de reconstruction $(x^T - PP^T x^T)^2$ entre le vrai x et sa reconstruction (après compression-décompression), $(PP^T x^T)^T = xPP^T$ (ici, la transformation $PP^T x^T$ correspond à la compression de x puis à sa décompression).
- TAL: on minimise la distance entre mot-cible y_o et son mot-input de contexte, x_c. Ici la distance est mesurée par un produit scalaire: $y_o W E^T x_c$ (et il y a un softmax qui traine, ok...).

Dans les 2 cas, on est passé par un espace de dimension plus petite, dans lequel les corrélations présentes entre les attributs de l'espace de départ sont clarifiées, factorisées en quelque sorte: dans le cas de la PCA, c'est très explicitement ce qu'on fait, puisque les vecteurs propres sont orthogonaux. Dans le cas du plongement pour le TAL (*word embedding*), c'est aussi l'idée: il y a des corrélations entre attributs (les mots du vocabulaire) dans les données (les corpus): par exemple deux synonymes apparaissent dans des contextes semblables. Dans l'esapce du plongement, ces synomnymes seront proches, au sens du produit scalaire, c'est-à-dire seront colinéaires. A contrario, des champs sémantiques différents (attributs-mots indépendants) seront éloignés dans le plongement, au sens du produit scalaire, c'est-à-dire seront a peu près orthogonaux.

De même que **la PCA est un pré-processing puissant qui permet de réduire la complexité d'un modèle** qui arrive après dans le pipeline, le plongement vectoriel permet de faire du TAL, là où l'espace du vocabulaire, de dimension V, ne permet pas de faire quoi que ce soit, si il est pris à l'état brut, car il n'a aucune structure (l'ordre des mots dans le dictionnaire ne signifie rien). L'idée est qu'en dimension V, il n'y a aucun rapport entre les mots proches tandis que dans D, les mots proches ont une proximité sémantique.

De plus, cette réduction de dimension permet de réduire l'overfitting. En effet, en réduisant la dimension de départ, on réduit le nombre de paramètres.

Bien que cette réduction dimensionnelle entraîne une perte d'informations (on fait de la compression avec perte), elle permet aussi de gagner de la structure. par exemple sur un jeu de données constitué d'images de dimension D de chats et de chiens, on aura juste des images tandis qu'on aura en dimension D' des données représentant des chats et des chiens.

21.1 [Hors programme] – Tout est probabilité

Quasiment[13] **toutes les choses qu'on a vues peuvent être exprimées comme des problèmes de probabilité**. Par exemple, **l'apprentissage supervisé revient toujours à chercher un modèle pour** $P(y|x,\theta)$.

L'apprentissage non supervisé est essentellement toujours de l'estimation de densité: on cherche à estimer $P(x|\theta)$, éventuellement, en K-means, on cherche le détail des $P(x_k|\theta_k)$, et la proba totale est la somme : $P(x|\theta) = \sum_k \pi_k P(x_k|\theta_k)$.

En TAL, on cherche clairement a avoir $P(y_o|x_c,\theta)$

De façon cruciale, on peut justifier la **méthodologie de train/validation/test** en **considèrant les données comme des réalisations de tirages aléatoires** de $x \sim P(x)$ (non supervisé) ou de tirage de paires $(x,y) \sim P(x,y)$. En effet, le but de l'apprentissage est de maximiser la performance de test. Le mini-raisonnement crucial est de dire que la distribution des données de test est estimée par la donnée des données d'entrainement et validation.

21.1.1 Calcul Bayésien: le Maximum A Posteriori (MAP), et la régularisation

Dans ce cadre, il y a un point important que nous n'avons pas eu le temps de voir en cours. Lors d'un calcul de maximum de vraisemblance (MLE), on a parfois une idée sur la valeur des paramètres, **a priori**, c'est-à-dire **avant d'avoir regardé les données**.

Prenons un cas techniquement très simple: le tirage de v.a. gaussiennes, dont on cherche à estimer les paramètres $\theta = (\mu, \sigma)$. On suppose, contrairement au calcul en MLE, qu'**on a une idée *a priori* sur la valeur de ces paramètres**, et en particulier sur μ, dont on pense qu'il est d'ordre de grandeur $\sim \tau$. Plus précisément, on a des raisons de penser (à cause du phénomène considéré, qu'on connait, ou bien de je ne sais quelle intuition) que la probabilité que la variable μ soit d'une certaine valeur μ est donnée par $\rho(\mu) = \mathcal{N}(0,\tau)$, c'est-à-dire que

$$\rho(\mu) = \frac{1}{\sqrt{2\pi\tau^2}} \exp\left(-\frac{(\mu)^2}{2\tau^2}\right) \tag{155}$$

C'est un peu méta: on a une intuition sur la probabilité que le paramètre μ prenne telle ou telle valeur, pourtant, **on ne pourra jamais répéter cette expérience**: la vraie valeur de μ, qu'on peut noter μ^*, est unique, c'est un certain nombre, et pas une loi, on ne pourra jamais "refaire l'expérience", et donc jamais vérifier le bien-fondé de notre a priori (la loi $\rho(\mu)$). C'est ce postulat un peu étrange (parler de la distribution de proba d'un truc dont on ne pourra jamais observer plus que 1 tirage) qui fait que **les fréquentistes n'aiment pas cette façon de faire les choses**.

On peut maintenant faire ce qu'on appelle un calcul Bayésien. Comme en MLE, on dispose de N tirages indépendants. Les notations sont les même que dans la section sur le MLE, sec. 17.1. Tous les points sont distribués selon la (même) loi Gaussienne:

$$\mathbb{P}(X_n \in [x, x+dx]) = \rho(x)\mathrm{d}x, \tag{156}$$

$$\rho(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{157}$$

Où $\theta = (\mu, \sigma)$ sont les 2 paramètres réels à déterminer (à "fitter", pour faire un anglicisme).

L'idée du **maximum a posteriori** (MAP) est de maximiser la vraisemblance des paramètres sachant les données (et non celle des données sachant les paramètres). On note quand même $\mathcal{L}$ (comme Likelihood, Vraisemblance en anglais):

$$\mathcal{L}(\theta) = \mathbb{P}(\theta|X = X_{(N,1)}) = \mathbb{P}(\theta|X) \quad \text{(notation informelle mais plus compacte)} \tag{158}$$

$$= \frac{\mathbb{P}(X|\theta)P(\theta)}{\mathbb{P}(X)} \tag{159}$$

$$\theta^* = \mathrm{argmax}_\theta(\mathcal{L}(\theta)) = \mathrm{argmax}_\theta \log(\mathcal{L}(\theta)) \tag{160}$$

$$= \mathrm{argmax}_\theta \left(\frac{\mathbb{P}(X|\theta)P(\theta)}{\mathbb{P}(X)}\right) \tag{161}$$

TODO: Francois Landes: ici je dois finir.

[13] Pour les arbres, ce n'est pas évident, et en fait ce n'est pas pratique de les voir de façon probabiliste, il me semble. Mais les *random forest* par contre, clairement oui.

[exemple de la regression linéaire comme MLE, qui explique le carré dans les moindres carrés: comme dans le DM]

[regression linéaire avec a priori sur les paramètres: explique la regularization]

[autre a priori: les classes dans un modeles baysien naif (loi uniforme)]

[autre a priori: les mots jamais vus dans un modele bayesien naif multinomial: mettre une petite proba non nulle]

[idem pour les pixels toujours éteins - mettre une petite proba non nulle]

[prior plus interessant: l'architecture meme du modele – ou verture philosophique, et vers le deep]

22 [Hors programme] Apprentissage non supervisé: Clustering

22.1 Algo des K-moyennes

On peut le voir comme une estimation de densité (ça n'est pas évident, à discuter plus tard). Essentiellement, on cherche à trouver des **groupes de points** (cluster) "proches" en un certain sens. Donc une **affectation** par point (on affecte un point à un groupe, et un groupe est constitué par l'ensemble des points qui y sont affectés).

Dans l'idée, ce clustering devrait satisfaire 2 conditions (simultanément):
- Le **représentant** du cluster se situe au point moyen (barycentre) **des points** de ce groupe
- **Un point** appartient au **cluster** dont le représentant est le plus proche

Sur le graphique (avec en gros 2 nuages de points, cf les diapos), par exemple, on peut imaginer que :
- abscisse = poids
- ordonnée = taille

Souvent on a plus que 2 features d'entrées (2 dimensions), mais c'est dur à représenter donc pas top niveau pédagogie.
Remarque: Faisons attention au choix de l'unité, choix qui influe sur la représentation des données.

22.2 Algo des k moyennes, pas à pas

Intuition : Pour chaque groupe (cluster), on aura un emplacement dans l'espace qu'on appelera **représentant** du groupe. On a des tas de points et on doit trouver des groupes tels que chaque représentant se situe au barycentre des autres points du groupe et qu'un **point** appartienne au groupe (cluster) dont le représentant est le plus proche.

Cas Typiques: le représentant peut être choisi comme la moyenne des points (on parle de leur barycentre, pour être précis dans les termes), ou bien comme la médiane des points (c'est alors l'algo des K-médoïdes).
Ici on fait les K-*moyennes* donc on utilise la *moyenne*. (K-means en anglais).

Détail de l'algo, pas à pas :
-(a) on place deux centres (au hasard).
-(b) on associe chaque point à un centre (selon la distance)
-(c) on recalcule les représentant des deux groupes créés: un représentant se trouve à la moyenne des points qui lui ont été assignés.
-(d) on repète les étapes (b-c-d), pour améliorer la position des représentants. On s'arrête après un nombre d'itérations maximum fixé à l'avance, ou bien parce que l'itération des étapes (b-c) ne change plus rien au résultat.

Pour les formules mathématiques de mise à jour des paramètres, voir la section suivante.

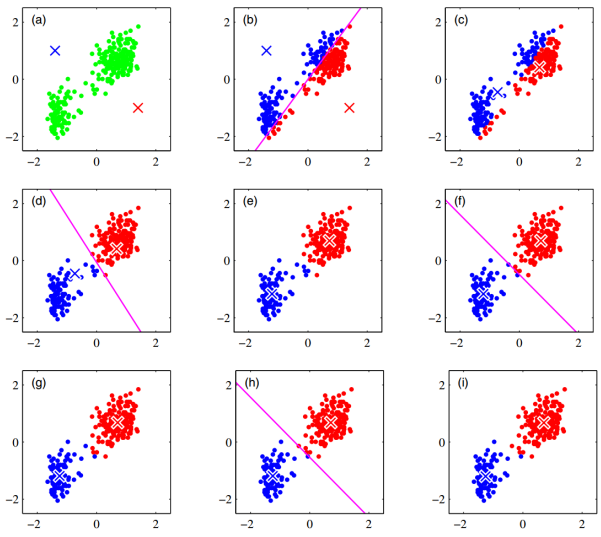


Figure 18: L'algo des K-moyennes, pas à pas. Tiré du livre de Bishop, page 426. Les étapes b,d,f,h sont équivalentes à l'étape (b) (associer le bon groupe à chaque point). Les étapes c,e,g,i sont équivalentes à l'étape (c) (recaluler le barycentre de chaque groupe, et déplacer le représentant du groupe en ce point).

Remarque non cruciale en théorie, mais qui peut poser des soucis en pratique:
Lorsque un représentant est initialisé a un endroit sans points, il faut gérer la situation en faisant un choix. On risque d'éliminer un cluster, si on fait certains (mauvais) choix. Pour cette raison, il peut être une bonne idée d'initaliser au hasard les affectations plutôt que les représentants. Ainsi, le problème d'avoir un groupe avec 0 points dedans à moins de chance de se produire. Cependant, cela peut tout de même arriver, et il faut prévoir ce cas dans le code, en toute rigueur (pour un petit exo de TP, ce n'est pas grave si on omet cela).

Question:
Comment choisir K ? On ne sait pas à l'avance, K est ce qu'on appelle un *hyper-paramètre*, c'est à dire qu'il n'est pas appris par l'algorithme, donc on choisit et on teste différentes valeurs.

22.3 Dérivation des étapes de l'algo des K-moyennes.

L'algo décrit dans la section précédente paraît marcher, intuitivement et en pratique, et on pourrait s'en tenir là, en constatant qu'on a "une recette qui marche bien".
Cependant, on est en droit de se demander: mais d'où ça sort ? Comment inventer des variantes de cet algo, mais de façon un peu cohérente, plutôt que de juste faire varier quelques choix au petit bonheur la chance ? Y-a t il une sorte de justification théorique au fait que cet algo, sans même l'implémenter et l'essayer, est *a priori* un "bon algo" ? (Pour des variantes qu'on pourrait vouloir inventer, avoir ce genre de savoir *a priori*, même si c'est un peu intuitif, serait bien pratique).

Dans cette section, on prouve que l'algo des K-moyennes peut être "dérivé" (comme on dérive une preuve) depuis le choix d'une fonction coût. L'algo découle ensuite de l'utilisation d'une stratégie classique d'optimisation de fonction coût.

La fonction coût qu'on souhaite miniser est définie ainsi:

$$J = \sum_{n}^{N} \sum_{k}^{K} a_{nk} |\overrightarrow{x}_n - \overrightarrow{\mu}_k|^2 \tag{162}$$

Intuitivement, les $\vec{\mu}_k$ sont les positions des représentants: par exemple $\vec{\mu}_1$ est un vecteur de D dimensions, qui est l'emplacement du centre du cluster numéro 1.
Intuitivement, les a_{nk} sont les affectations. On peut imaginer qu'elles valent 1 quand le point n est affeté au groupe k, et 0 sinon. Par exemple $a_{12,3} = 1$ indique que le point $n = 12$ est affecté au groupe $k = 3$.

Justification de J: Si on réfléchit un peu sur cette définition de J, on voit que c'est la somme des carrés des distances entre les points de chaque cluster et le représentant du groupe où ces points ont été affectés. Il est clair qu'un bon clustering devrait correspondre à ce que ce J soit plutôt petit.

Stratégie d'optimisation

Quand doit optimiser une fonction coût J qui est une fonction de deux variables très distinctes en nature (ici, les $\vec{\mu}_k \in \mathbb{R}^D$ et les $a_{nk} \in \{0, 1\}$), une stratégie classique est d'itérer 2 étapes de minimisation, jusqu'à convergence (où avant, si il n'y a pas convergence).

Concrètement, on fixe la valeur d'une des deux variables (par exemple on fixe les a_{nk}) et on optimise $J(\vec{\mu}_k)$. On trouve ainsi un $\vec{\mu}_k^*$ qui réalise un minimum de J.

L'étape suivante est alors de faire la même chose pour l'autre variable, ici les a_{nk}: on fixe la valeur de $\vec{\mu}_k$ et on cherche le meilleur a_{nk} possible, noté a_{nk}^*.

On itère ces deux étapes autant qu'on le peut, ou jusqu'à convergence.

Cette stratégie a particulièrement de sens lorsqu'on peut calculer le minimum global de $J(\vec{\mu}_k)$ d'un coup.

22.3.1 Re-calcul des représentants

On commencer par fixer les a_{nk} et on optimise $J(\vec{\mu}_k)$. On va ainsi trouver un $\vec{\mu}_k^*$ qui réalise un minimum de J.

Pour trouver un minimum d'une fonction, on utilise la méthode de la dérivée, les points de max et min ont une dérivée nulle. C'est pareil avec le gradient pour une fonction de plusieurs variables: les points de min ou max out un gradient nul, $\vec{\nabla} J = \vec{0}$.

On prend un k' générique, et on dérive J par rapport à ce vecteur $\mu_{k'}$ (on prend le gradient quoi). On cherche à résoudre $\vec{\nabla} J = \vec{0}$.

$$J = \sum_n^N \sum_k^K a_{nk} |\overrightarrow{x}_n - \overrightarrow{\mu}_k|^2 \tag{163}$$

$$\vec{\nabla}_{\vec{\mu}_{k'}} J = \vec{\nabla}_{\vec{\mu}_{k'}} \sum_n^N \left(\left(\sum_{k \neq k'}^K a_{nk} (\overrightarrow{x}_n - \overrightarrow{\mu}_k)^2 \right) + a_{nk'} (\overrightarrow{x}_n - \overrightarrow{\mu}_{k'})^2 \right) \tag{164}$$

$$= \sum_n^N \left(0 + a_{nk'} \vec{\nabla}_{\vec{\mu}_{k'}} (\vec{x}_n - \vec{\mu}_{k'})^2 \right) \quad \text{car } \mu_{k'} \text{ n'apparaît pas dans le terme de gauche} \tag{165}$$

On développe le produit scalaire $(\vec{x}_n - \vec{\mu}_{k'})^2$ et le gradient $\vec{\nabla}$ explicitement: (166)

$$= \sum_{n=1}^N a_{nk'} \begin{pmatrix} \frac{\partial \cdot}{\partial \mu_{k'1}} \\ \vdots \\ \frac{\partial \cdot}{\partial \mu_{k'D}} \end{pmatrix} [(x_{n1} - \mu_{k'1})^2 + (x_{n2} - \mu_{k'2})^2 + \cdots + (x_{nD} - \mu_{k'D})^2] \tag{167}$$

$$= \sum_{n=1}^N a_{nk'} \begin{pmatrix} 2(x_{n1} - \mu_{k'1})(-1) + 0 + \ldots + 0 \\ 0 + 2(x_{n2} - \mu_{k'2})(-1) + 0 + \ldots + 0 \\ \vdots \\ 0 + \ldots + 0 + 2(x_{nD} - \mu_{k'D})(-1) \end{pmatrix} \quad \text{On a utilisé } (u(x)^n)' = n.u'(x).(u(x))^{n-1} \tag{168}$$

$$= -2 \sum_n^N a_{nk'} (\overrightarrow{x_n} - \overrightarrow{\mu_{k'}}) \tag{169}$$

La dernière formule s'obtient par identification. Lorsqu'on est entraîné.e, on peut quasiment le faire d'un coup, à l'oeil nu. On cherche maintenant à résoudre $\vec{\nabla}_{\vec{\mu}_{k'}} J = \vec{0}$:

(170)

$$\vec{0} = \vec{\nabla}_{\vec{\mu}_{k'}} J \tag{171}$$

$$\vec{0} = \sum_n^N a_{nk'} \overrightarrow{x_n} - \sum_n^N a_{nk'} \overrightarrow{\mu_{k'}} \tag{172}$$

$$\iff \sum_n^N a_{nk'} \overrightarrow{x_n} = \sum_n^N a_{nk'} \overrightarrow{\mu_{k'}} \tag{173}$$

$$\iff \overrightarrow{\mu_{k'}} = \frac{\sum_n^N a_{nk'} \overrightarrow{x_n}}{\sum_n^N a_{nk'}} \tag{174}$$

$\vec{\mu}_{k'}$ est clairement le barycentre des points $\vec{x}_n$ qui sont dans le cluster k'

22.3.2 Ré-assignation des points

On fixe maintenant les $\vec{\mu}_k$ et on optimise $J(a_{nk})$. On va ainsi trouver une affectation a^*_{nk} qui réalise un minimum de J (en fait là on parle de la matrice complète de a_{nk}, c'est une matrice de taille N, K).

On cherche le minimum de J, mais ici les variables a_{nk} sont discrètes et non pas continues, donc on ne peut pas faire de l'optimisation continue (par exemple, calculer la dérivée). À la place, on fait de l'optimisation combinatoire (optimisation sur des objets discrets, quoi). Mais en pratique, ici, c'est très simple, il n'y a pas tellement de combinaisons possibles.

Pour $\vec{a}_n$ il y a K possibilités (tous les vecteurs possibles avec un 1 à la k-ème place, donc il y en a K différents). On va s'intéresser à une variable $\vec{a}_{n'}$ en particulier, pour un n' fixé.

$$J = \sum_n^N \sum_k^K a_{nk} |\overrightarrow{x}_n - \overrightarrow{\mu}_k|^2 \tag{175}$$

$$J = \sum^K a_{n'k} (\vec{x}_{n'} - \vec{\mu}_k)^2 + \sum_{n \neq n'}^N \sum_a^K a_n^k (\vec{x}_n - \vec{\mu}_k)^2 \tag{176}$$

On a isolé le morceau de J qui dépend de $\vec{a}_{n'}$ (premier terme), et l'autre bout, qui n'en dépend pas. Tous les nombres du termes de droites sont indépendant du choix de $\vec{a}_{n'}$, de son point de vue, ces termes sont donc constants.

On développe maintenant la somme du premier terme:

$$J = \sum_{k=1}^{K} a_{n'k}(\vec{x}_{n'} - \vec{\mu}_k)^2 \tag{177}$$

$$= a_{n'1}(\vec{x}_{n'} - \vec{\mu}_1)^2 + a_{n'1}(\vec{x}_{n'} - \vec{\mu}_2)^2 + \ldots + a_{n'K}(\vec{x}_{n'} - \vec{\mu}_K)^2 + \text{const.} \tag{178}$$

Comme on sait que parmi tous les $a_{n'k}, k = 1 \ldots K$, il faut mettre à 1 une seule valeur, et les autres à 0, on voit que dans cette équation avec K termes, il faut mettre $a_{n'k} = 1$ pour le k tel que $(\vec{x}_{n'} - \vec{\mu}_k)^2$ soit le plus petit:

$$k^* = \text{“le meilleur choix d'affectation du point } x_n \text{ à un cluster”} \tag{179}$$

$$= \text{argmin}_{k \in [|1,K|]}(\vec{x}_{n'} - \vec{\mu}_k)^2) \tag{180}$$

22.4 Convergence ?

On ne discute pas trop les conditions de convergence. On espère et on a de bonnes raisons de croire que ça converge. Une raison d'y croire est que clairement, par ce processus itératif, on diminue J à chaque itération, et si on atteint un point où plus rien ne se passe, c'est qu'on a atteint un minimum (local, au pire).

22.5 Affectations molles

La position d'un centre que l'on peut aussi nommer représentant est le centre (ou la moyenne) des points qui lui sont affectés. Dès lors, il existe deux types d'affectations :

- L'affectation dure consiste en le fait d'affecter chaque donnée à un et un seul centre (le centre le plus proche de ce point).
- L'affectation “molle” consiste en le fait d'attribuer à chaque donnée une probabilité d'appartenir à chaque centre.

22.6 Pseudo-code de l'algo des K-moyennes

Les hyper-paramètres et autres variables sont:
N, la taille des données d'apprentissage.
D, la dimension des données
K, le nombre de clusters que l'on souhaite définir, trouver.

Les structures de données sont:
$X_{(N,D)}$: les données d'entrainement.
affect$_{(N,K)}$: les affectations (du point n au cluster k, si $affect(n,k) = 1$)
centres$_{(K,D)}$: les coordonnées des centres (représentants).

Part IV
Autres exercices - hors programme

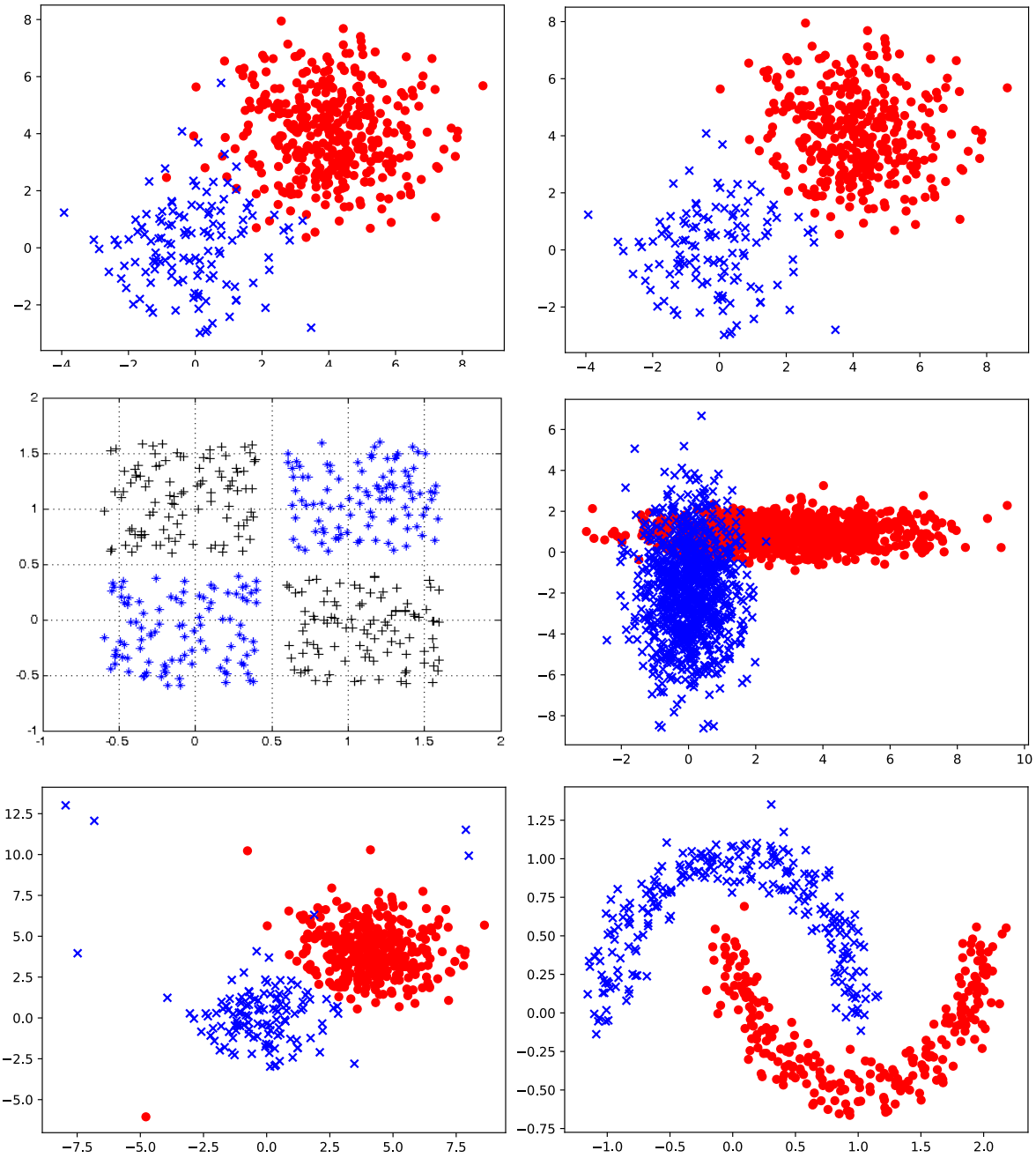
23 Bonus – Autres exercices (pas forcément abordés en classe)

23.1 Données à problèmes

On vous présente des jeux de données étiquetées binaires (2 classes), qui, coup de chance, ne sont qu'en deux dimensions, ce qui les rend représentables sur papier.

Algorithm 2 K moyennes

procedure KMOYENNES($X, MaxIter$)
  Centres = CentresInitiaux(K,D) ▷ On suppose qu'on a un moyen d'avoir des centres initiaux
  for *iter* in [1,...,MaxIter] **do**
    for n in [1,...,N] **do** ▷ Mise à jour des affectations
      distancesPointCentres = $|X[n] - \text{centres}[:]|$
      $\forall k, \text{affect}[n,k] = 0$
      $k = argmin_{k'}(\text{distancesPointCentres}[k'])$
      $\text{affect}[n,k] = 1$
    for k in [1,...,K] **do** ▷ Mise à jour des positions des centres
      $\text{centres}[k] = \text{affect}[:,k].X[:,:]$ ▷ Produit matriciel (contraction de 1 indice)
      NbpointDansGroupek = $sum(\text{affect}[:,k])$
      if NbPointDansGroupek ==0 **then**
        $\text{centres}[k] = \text{centres}[k] * 0$ ▷ Ce n'est pas idéal, à discuter en cours
      else
        $\text{centres}[k] = \text{centres}[k]/$ NbPointDansGroupek
  return centres, affect



Pour chaque jeu de données, par observation, dire si il est:

- linéairement séparable, ou pas
- un bon candidat pour un classificateur linéaire
- séparable avec un noyau, ou pas
- un bon candidat pour un classificateur avec noyau
- si il n'est pas séparable de ces façon là, expliquer quel est le "problème" du jeu de données, et si il peut être "résolu"

Remarque – le code qui génère ces données est disponible, vous pouvez l'utiliser pour tester des algos sur ces données: `https://gitlab.inria.fr/flandes/ias/-/raw/master/TP-sujets/TD-Bonus-generateur-de-blobs.py?inline=false`

23.2 Algo de type "pas vus en cours mais faisable": Classifieur à distance minimum

Crédit: Claude Barras

Un classifieur à distance minimum fonctionne en décidant d'associer à un point (du *test set*) la classe dont le représentant est le plus proche. Pour minimiser l'erreur sur l'ensemble d'apprentissage, il faut que le représentant d'une classe soit le barycentre (la moyenne) des points de cette classe (sur les données d'apprentissage). C'est donc un apprentissage *one shot*, il se fait "d'un coup".

Dans cet exercice, on se limite à deux classes dans un espace vectoriel de dimension 2 muni du produit scalaire (et donc de la distance euclidienne).

1. Expliciter la logique de l'algorithme: quels sont ses paramètres, combien sont ils, comment les entraîne t-on ? Quels sont les hyper-paramètres ?

2. Expliciter la logique de l'algorithme: comment produit on la prédiction y pour la classe d'une nouvelle donnée x_{test} qui n'a pas été observée lors de l'apprentissage ?

3. Écrire le pseudo code de l'algorithme. On précisera bien la taille des vecteurs et matrices utilisés, en indiquant le sens de chacune des variables. On tâchera de choisir des noms d'indices parlants (par exemple $d = 1, 2, \ldots, D$)

4. On dispose des quatre jeux de données suivants.

Données 1 :
$$X_{-1} = \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right\}$$
$$X_{+1} = \left\{ \begin{pmatrix} 2 \\ 0 \end{pmatrix}, \begin{pmatrix} 2 \\ 1 \end{pmatrix}, \begin{pmatrix} 3 \\ 0 \end{pmatrix}, \begin{pmatrix} 3 \\ 1 \end{pmatrix} \right\}$$

Données 2 :
$$X_{-1} = \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right\}$$
$$X_{+1} = \left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 2 \\ 0 \end{pmatrix}, \begin{pmatrix} 2 \\ 1 \end{pmatrix}, \begin{pmatrix} 3 \\ 0 \end{pmatrix}, \begin{pmatrix} 3 \\ 1 \end{pmatrix}, \begin{pmatrix} 4 \\ 0 \end{pmatrix}, \begin{pmatrix} 4 \\ 1 \end{pmatrix} \right\}$$

Données 3 :
$$X_{-1} = \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 2 \\ 0 \end{pmatrix}, \begin{pmatrix} 2 \\ 1 \end{pmatrix}, \begin{pmatrix} 3 \\ 0 \end{pmatrix}, \begin{pmatrix} 3 \\ 1 \end{pmatrix}, \begin{pmatrix} 4 \\ 0 \end{pmatrix}, \begin{pmatrix} 4 \\ 1 \end{pmatrix} \right\}$$
$$X_{+1} = \left\{ \begin{pmatrix} 2 \\ 2 \end{pmatrix}, \begin{pmatrix} 2 \\ 3 \end{pmatrix}, \begin{pmatrix} 3 \\ 2 \end{pmatrix}, \begin{pmatrix} 3 \\ 3 \end{pmatrix}, \begin{pmatrix} 4 \\ 2 \end{pmatrix}, \begin{pmatrix} 4 \\ 3 \end{pmatrix}, \begin{pmatrix} 5 \\ 2 \end{pmatrix}, \begin{pmatrix} 5 \\ 3 \end{pmatrix}, \begin{pmatrix} 6 \\ 2 \end{pmatrix}, \begin{pmatrix} 6 \\ 3 \end{pmatrix} \right\}$$

Données 4 :
$$X_{-1} = \left\{ \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \begin{pmatrix} 2 \\ 1 \end{pmatrix}, \begin{pmatrix} 2 \\ 2 \end{pmatrix} \right\}$$
$$X_{+1} = \left\{ \begin{pmatrix} 0 \\ 3 \end{pmatrix}, \begin{pmatrix} 1 \\ 3 \end{pmatrix}, \begin{pmatrix} 2 \\ 3 \end{pmatrix}, \begin{pmatrix} 3 \\ 2 \end{pmatrix}, \begin{pmatrix} 3 \\ 1 \end{pmatrix}, \begin{pmatrix} 3 \\ 0 \end{pmatrix} \right\}$$

Pour chacun de ces jeux de données:

4.a Calculer un représentant de chaque classe (pour ce modèle, c'est le barycentre des points de la classe)

4.b Calculer l'équation de la surface de décision et en déduire la règle de décision

4.c Dire si des points d'apprentissage sont mal classés.

5. Implémenter tout ça en python (sur au moins 2 des jeux de données). Visualiser les résultats.

23.3 Regression Linéaire - version Algèbrique

Plutôt que de faire une descente de Gradient, on peut calculer la régression linéaire d'un coup. On considère la minimisatin de la fonction coût $J = \frac{1}{2}\frac{1}{N}\sum_n^N(\vec{\theta}\cdot\vec{x}_n - y_n)^2$.

1. Rappeler l'expression de $\vec{\nabla}_{\vec{\theta}}J$.

2. Ré-écrivez cette expression sous la forme la plus matricielle possible (donc sans somme). Pensez à écrire comme on écrit les calculs en numpy: matrices et produits de matrices, pas de somme. Indiquez la taille des matrices au pied de chaque matrice : par exemple, $X_{(N,D)}$ ou encore $\vec{\theta}_{(D,1)}$. Aide : le produit scalaire se réécrit: $\vec{a}_{(D,1)}\cdot\vec{b}_{(D,1)} = (\vec{a}^T)_{(1,D)}.\vec{b}_{(D,1)} = resultat_{(1,1)} \in \mathbb{R}$

3. Comme J est convexe en $\vec{\theta}$, il y aura un seul extremum, qui sera aussi un minimum global (on ne détaille pas ce raisonnement là). Quelle est donc l'équation qui permet de trouver les paramètres optimaux $\vec{\theta}^*$ qui minimisent J ? (il n'y a pas de piège !)

4. En supposant que les matrices qu'on a besoin d'inverser sont en effet inversibles, "calculez" la solution θ^* (concrètement, il faudrait évidemment avoir les données, et il y aurait une grosse grosse matrice à inverser, donc restez au stade des calculs abstraits dans cette question).

5. Bravo, vous avez dérivé la solution exacte de la régression linéaire !

6. Vous pouvez faire de même pour la fonction coût régularisée: $J = \frac{1}{2}\vec{\theta}^2 + \frac{1}{2}\frac{1}{N}\sum_n^N(\vec{\theta}\cdot\vec{x}_n - y_n)^2$.

23.4 Entropie – papier-crayon

L'entropie est un concept qui intervient dans l'algorithme d'apprentissage des arbres (de décision ou de régression).

Voir le 2.4.5 du poly de maths de L2 pour un rappel de la définition de l'entropie.

1. Calculez l'entropie de la variable aléatoire associée au lancé de 1 dé (c'est la v.a. qui au lancé de dé associe la valeur observée sur le dessus du dé).

2. Calculez l'entropie de la variable aléatoire associée au lancé de 2 dés indépendants (c'est la v.a. qui au lancé de dé associe le couple de valeurs observées sur le dessus des dés).

23.5 Bayésien Naïf – modèle Gaussien

1. Écrire le pseudo-code du modèle Bayésien Naïf. On choisit un modèle Gaussien. Bien dénombrer les paramètres qui seront à "apprendre".

2. Implémenter ce code en python.

3. L'appliquer aux données de l'exercice 23.1 (1ère et 4ème figure en particulier), visualiser le résultat. En quoi ce modèle est il différent du calssifieur à distance minimale (exercice 23.2)

4. L'appliquer aux données MNIST (apprentissage supervisé des 10 classes). Tester voter modèle, en utilisant la méthodologie appropriée. Quel est la performance obtenue ?

5. Visualisez les paramètres appris sous forme d'images en noir et blanc. On regroupera les paramètres correspondant à chaque classe.

6. En quoi le jeu de données MNIST est il critiquable (si on le compare à des jeux de données comme CIFAR-10, CIFAR-100 ou ImageNet) ? Que peut-on en déduire sur la performance attendue de ce genre de modèles, pour des images du genre CIFAR-10, etc ?

23.6 Bayésien Moins Naïf – modèle Gaussien

Réfléchir à l'extension du modèle Bayésien naïf au cas où on considère les features comme corrélées par une matrice de covariance fixée, c.a.d. que le modèle Gaussien choisi a une matrice de covariance non diagonale.

Quel est le lien avec un modèle naïf auquel on appliquerait une *feature map* polynomiale d'ordre 2 ?

Quel serait l'apport d'une PCA en pre-processing d'un tel modèle ? Pensez vous qu'il serait mieux d'utiliser une PCA, même avec beaucoup de composantes préservées, ou bien qu'il vaut mieux faire sans ?

Implémentez ce modèle, et appliquez le par exemple à la classification MNIST.

23.7 Algo pas vu en cours: Algorithme des k-moyennes

En 2020-21, on avait vu cet algo en cours. Ici, cet exercice présente cet algorithme de façon auto-suffisante (mais sans rentrer dans la théorie).

On considère un espace vectoriel de dimension 2 muni du produit scalaire canonique (et donc de la distance Euclidienne). On dispose des points d'apprentissage suivants :

$$x_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, x_2 = \begin{pmatrix} 2 \\ 2 \end{pmatrix}, x_3 = \begin{pmatrix} 2 \\ 4 \end{pmatrix}, x_4 = \begin{pmatrix} 4 \\ 0 \end{pmatrix}, x_5 = \begin{pmatrix} 4 \\ 2 \end{pmatrix}, x_6 = \begin{pmatrix} 6 \\ 1 \end{pmatrix}, x_7 = \begin{pmatrix} 0 \\ 2 \end{pmatrix}$$

1. Exécuter trois itérations de l'algorithme k-moyennes en utilisant comme centre initiaux (représentants initiaux): $r_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, r_2 = \begin{pmatrix} 2 \\ 2 \end{pmatrix}$

A chaque itération,

- Indiquer pour chaque exemple quelle est le centre le plus proche (la distance exacte n'est pas demandée),
- Indiquer les coordonnées des nouveaux centres après recalcul des moyennes,
- Tracer dans une figure les exemples et les deux centres.

2. Écrire le pseudo code de l'algorithme ci dessus. Si possible, formater le tableau des affectations avec autant de colonnes que de groupes (clusters). Cela permet de facilement généraliser votre code au cas des affectations molles (avec *softmax*). On précisera bien la taille des vecteurs et matrices utilisés, de cette façon: $X_{N,D}$ en indiquant le sens de chacune des variables. On tâchera de choisir des noms d'indices parlants (par exemple $d = 1, 2, \ldots, D$)

3.1 Implémenter l'algorithme en python, le faire tourner et vérifier ce qui a été vu sur papier. [Si vous avez des bogues difficiles à cerner, vous pouvez tracer les points (colorés par groupe d'appartenance), les représentants (*marker* plus gros et coloré correctement) et éventuellement les frontières de décision (médiatrices entre les représentants), afin de vous aider à voir ce qui n'est pas correct.]

3.2. Calculer un critère de qualité et en faire le suivi au cours des itérations (époques).

3.3. Partir de représentants initiaux r_1, r_2 aléatoires, et observer les variations des résultats en fonction de cette condition initiale.

3.4. Tester votre code sur les données de l'exercice 1.4 (1ère et 4ème figure en particulier), dont la génération est faite pas un code python disponible en ligne. Pensez bien à respecter la méthodologie de test appropriée.

23.8 Bonus – Régression Linéaire interprétée comme du MLE

On souhaite maintenant comprendre comment la Régression Linéaire fonctionne, ou plus précisément on cherche à justifier la forme de la fonction coût à minimiser. On ne va pas s'intéresser au terme de régularisation pour le moment (il sera justifié plus tard, en M1). On cherche à justifier qu'il faut minimiser la fonction coût suivante:

$$J(X, Y, \Theta) = \sum_n (\vec{x}_n \vec{w} - y_n)^2$$

où $X = (\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N)$, $Y = (y_1, y_2, \ldots, y_N)$ représentent les données dans leur ensemble.

On choisit de modéliser les relations entre les x_n et les y_n de la façon suivante:

$$y_n = \vec{w}^T \vec{x}_n + \varepsilon,$$

où ε est un bruit gaussien centré: $\varepsilon \sim \mathcal{N}(0, \sigma^2)$. Autrement dit, la loi de la variable aléatoire $y_n - \vec{w}^T \vec{x}_n$ est supposée être celle de $\varepsilon \sim \mathcal{N}(0, \sigma^2)$.

La probabilité d'observer un couple de $(\vec{x}_n, y_n)$ donné, à $\vec{w}$ fixé (inconnu, mais fixé), est donc:

$$\rho(\vec{x} = \vec{x}_n, y = y_n) | \vec{w}) = \frac{1}{\sqrt{2\pi\sigma^2}} exp\left(-\frac{1}{2\sigma^2}(y_n - \vec{w}^T \vec{x}_n)^2)\right)$$

En utilisant l'idée de l'estimation du maximum de vraisemblance, finir le raisonnement et en conclure que la fonction à minimiser est de la forme $J(X, Y, \Theta) = \sum_n (\vec{x}_n \vec{w} - y_n)^2$ (sachant que le seul paramètre à optimiser est le vecteur $\vec{w}$).

Aide: il n'y a vraiment pas grand chose à faire ! Si vous voulez, vous pouvez même calculer la solution analytique, qui permet de trouver directement la solution.

23.9 Révisions d'algèbre – Valeurs propres - Limite de suite - Fibonacci ‡

Corrigé: cf 27.15.

La suite de Fibonacci est définie ainsi : $F(0) = 0, F(1) = 1, F(n) = F(n-1) + F(n-2)$. En notant $\vec{x}_n = \begin{pmatrix} F(n) \\ F(n-1) \end{pmatrix}$ l'état de la suite sous forme vectorielle, on va simplifier le calcul du terme $F(n)$.

1. Écrire $\vec{x}_1$.
2. Trouver la matrice T telle que $\vec{x}_{n+1} = T \cdot \vec{x}_n$. (Pour commencer, quelle est la dimension de T? Puis, chercher la première ligne, qui permet de calculer la première composante de $\vec{x}_{n+1}$. Puis, la deuxième ligne.)
3. Vérifiez que votre T marche sur quelques itérations, en calculant $F(2), F(3)$ à l'aide de cette matrice.
4. On rentre cette matrice dans un ordinateur (on peut aussi le faire à la main), et on trouve qu'elle est diagonalisable, avec comme système propre:

$$\lambda_1 = \frac{1+\sqrt{5}}{2}, \vec{v}_1 = \begin{pmatrix} \frac{1+\sqrt{5}}{2} \\ 1 \end{pmatrix}$$
$$\lambda_2 = \frac{1-\sqrt{5}}{2}, \vec{v}_2 = \begin{pmatrix} \frac{1-\sqrt{5}}{2} \\ 1 \end{pmatrix}$$

   Expliciter la valeur de la matrice de passage P et de la matrice diagonale des valeurs propres D (aussi parfois notée Λ). Il n'est pas nécéssaire de calculer P^{-1}. Donner l'expression de T en fonction de D, P, P^{-1} (c'est du cours).
5. Écrire maintenant directement $\vec{x}_n$ en fonction de la matrice diagonale des valeurs propre D, de la matrice de passage P et de $\vec{x}_1$.
6. (Question Bonus) On rappelle que $\sqrt{5} \approx 2.2$. Dans la limite de $n \to \infty$, quel sera le comportement asymptotique de $F(n)$? (indice: chercher plutôt le comportement asymptotique de $\vec{x}_n$). Pour aider un calcul explicite, on donne le résultat de la multiplication: $P^{-1}\vec{x}_1 = \begin{pmatrix} \frac{1}{\sqrt{5}} \\ -\frac{1}{\sqrt{5}} \end{pmatrix}$

23.10 Révisions d'algèbre – Valeurs propres et vecteurs propres - Mise en équation et étude de stabilité

Cet exercice est plus intéressant à faire avec un ordinateur, en choisissant des valeurs pour les coefficients (ou directement pour les valeurs propres).

On suppose que la production de biens et services se divise en 3 secteurs seulement: la production de biens (manufacturés), la production de services, et l'exploitation de ressources naturelles. La production annuelle de l'année n, notée $\vec{x}_n$, est un vecteur $(p_1, p_2, p_3)^T$. On s'intéresse à la variation de production (croissance où décroissance) d'une année sur l'autre. On suppose que :

- La production de biens (de l'année suivante) dépend de celle de l'année précédente, et de l'extraction de matières premières: $p'_1 = c_1 p_1 + c_{1,3} p_3$
- La production de services varie ainsi: $p'_2 = c_2 p_2 + c_{2,1} p_1$
- L'extraction de ressources varie ainsi: $p'_3 = c_{3,1} p_1 - c_3 p_3$

1. Quel est la matrice A qui vérifie: $\vec{x}_{n+1} = A\vec{x}_n$?
2. Calculer la production annuelle après 2 ans, en calculant le carré de A.
3. Comment calculer efficacement la production sur les années suivantes ? (indice: penser au système de valeurs propres/vecteurs propres). Exprimez le (sans tout calculer, ici on ne connait pas les coefficients de toute façon).
4. On suppose que les coefficients $c_1, c_2, c_3, c_{1,3}, c_{2,1}, c3, 1$ sont tels que A est diagonalisable. On suppose que les trois paires $\lambda_k, \vec{v}_k$ sont connues. De plus, on choisit de trier les valeurs propres de telle sorte que $\lambda_1 > \lambda_2 \geq \lambda_3$ (pour simplifier, on se place dans le cas où $\lambda_1 > \lambda_2$, strictement). Quel sera le comportement de $\vec{x}_n$ à grand n, selon la valeur de λ ? En particulier, discuter les cas $0 < \lambda_1 < 1$ et $1 < \lambda_1$.
5. Question Bonus. Que se passe-t il dans le cas $\lambda_1 = 1$?

24 Bonus – TP – Estimation de densité vs histogrammes

Le TP est téléchargeable ici: `https://gitlab.inria.fr/flandes/ias/-/raw/master/TP-sujets/TP-Bonus-histogra`
`zip?inline=false`

25 Bonus – SVM

25.1 SVM: quelques notions

Un modèle SVM (Support Vector Machine) linéaire se réduit à un simple classificateur linéaire $h(x)$ (Linear Discriminant Analysis) parametré par $\vec{w}$ et b, qui sépare l'espace par un hyperplan d'équation $h(\vec{x}) = \vec{w}\cdot\vec{x}+b = 0$. L'idée est de classifier comme avec un perceptron, mais en maximisant la marge (*margin*), c.a.d. la distance entre l'hyperplan et les points qui en sont le plus proche (et si ils sont mal classés, minimiser la distance à l'hyperplan).

Formellement, avec un ensemble d'apprentissage $(\vec{x}_n, y_n)_{n=1}^N$ de taille N, avec $\vec{x}_n \in \mathbb{R}^D$ et $y_n \in \{-1, 1\}$, on cherche à trouver $\vec{w}, b$ tels que:

$$\text{minimiser } \frac{1}{2}\ ||\vec{w}||^2 + C\sum_{i=n}^{N} \zeta_i \tag{181}$$

$$\text{sous les constraintes } \forall n, \quad y_n(\vec{w}\cdot\vec{x}_n + b) \geq 1 - \zeta_n \tag{182}$$

$$\forall n, \quad \zeta_n \geq 0 \tag{183}$$

La solution s'obtient sous la forme $\mathbf{w} = \sum_{i=n}^{N} \alpha_n y_n \mathbf{x_n}$, avec une série de coefficients $(\alpha_n)_{n=1}^N, \alpha_n \geq 0$ à déterminer.

Les points d'entrée $\mathbf{x_n}$ tels que $\alpha_n > 0$ sont appellés vecteurs supports (*support vectors*).

Les variables ζ_n sont les *slack variables*, elles permettent de "donner du mou" à la solution, c.a.d. autorisent que des points soient mal classés. Plus précisément, les points correctement classifiés ou sur la marge ont $\zeta_n = 0$, les points situés dans la marge (la zone intermédiaire entre les vecteurs supports et la frontière) ont $0 < \zeta_n \leq 1$, et les points mal classés ont $\zeta_n > 1$.

25.2 SVM linéaire: prise en main

25.2.1 Cas sans bruit (linéairement séparable)

1. Téléchargez le fichier "TP2-SVM-carré-énoncé.py".
2. À l'aide de la fonction *make_separable_square*, créez un jeu de données d'entraînement, et indépendamment, un jeu de données de validation.
3. Définissez et entraînez un SVM linéaire (lisez d'abord bien la doc. de *sklearn.svm.SVC*)
4. L'affichage du score à déjà été prévu. En gros, dans *sci-kit learn*, quasiment tous les modèles ont une méthode *fit* et une méthode *predict*, parfois assorties d'une méthode *score* (qui peut être trompeuse, par exemple si les classes ne sont pas équilibrées). À ce stade il se peut que le score d'entraînement et de validation soient très proches.
5. Représenter les vecteurs supports dans le plan (à nouveau, lisez la doc. de *sklearn.svm.SVC* pour savoir comment les récupérer).

25.2.2 Cas avec bruit (non linéairement séparable)

Ce cas est aussi un premier cas d'optimisation d'un hyper-paramètre.

1. Dans le même fichier, laissez vous guider par le TODO. Plus précisément:
2. Modifiez *make_noisy_square()* de sorte à ajouter du bruit, sous la forme d'une inversion des labels ($0 \to 1, 1 \to 0$) pour une fraction $p = 0.1$ des points (prise aléatoirement).
3. Dans un premier temps, entraînez le modèle pour une valeur fixée de C, tracez les résultats, récupérez le score etc, pour voir si tout fonctionne à peu près.

4. Faites varier l'hyper-paramètre C (dont vous devez avoir compris le rôle), et tracer le résultat en fonction de C.

5. En particulier, dans un graphique final, tracer les performances (train et validation) en fonction de C. Quel serait le choix de valeur que vous privilégieriez ? Astuce: la fonction *plt.semliogx* permet de faire un plot en echelle semi-logarithmique

25.3 SVM avec noyau

25.3.1 Jeu de données des demi-lunes

1. Générez et visualisez un jeu d'entraînement et de validation (utilisez le fichier "TP2-SVM-Moons-énoncé.py").

2. Entraînez un modèle linéaire, puis polynomial avec un C d'abord fixé.

3. Dans le cas polynomial en particulier, faites varier C, observez l'évolution des performances, et l'aspect du modèle appris (c.a.d. l'aspect de la frontière de décision en fonction de C).

4. Observez l'effet de C sur la frontière de décision. Désormais le rôle de C devrait vous apparaître plus clairement.

5. Bravo, vous avez optimisé votre premier hyper-paramètre dans un cas non trivial ! Mettez le code dans une fonction, de sorte à pouvoir aisément faire varier un ou plusieurs autres hyper-paramètres (par exemple le choix du type de noyau).

6. Bonus: essayer d'autres noyaux.

25.3.2 MNIST

N'ayons pas peur des grandes dimensions:

1. utilisez le fichier "TP2-SVM-MNIST-énoncé.py").

2. Classifiez MNIST à l'aide d'un SVM à noyau. Pensez à bien respecter la méthodologie de travail du Machine Learning.

3. Débrouillez-vous ! Hé oui maintenant vous êtes grands, vous êtes autonomes !

26 Bonus – Algo EM

26.1 [Hors programme] Algorithme EM

Dans ce TP, on découvre par l'exemple ce que c'est que l'algo EM, en partant du modèle BAyésien Naïf (supervisé), puis en effaçant les labels.

Pour celles/ceux qui ont déjà terminé le modèle Bayésien Naïf (chez eux ou pendant la séance), vous pouvez passer à cet exercice, qui consiste à faire de l'apprentissage non supervisé sur MNIST.

1. Téléchargez le fichier "TP-Bonus-algo-EM-MNIST-enonce.py".

2. Complétez le programme, en vous aidant des endroits indiqués "TODO" et du sujet de TD 2.3, ainsi que des notes de cours. (sur ce TP, on vous guide moins).

3. Astuce: pour savoir si votre algorithme converge/a convergé, vous pouvez suivre la valeur de la vraisemblance (ou mieux, la log-vraisemblance) au cours des époques (itérations) de l'algo EM. Si vous atteignez une constante, c'est que vous avez probablement convergé.

4. Bonus: Une fois que tout fonctionne bien, vous pouvez éventuellement charger le jeu de données MNIST complet, c.a.d. en résolution 28×28 (contenu dans le fichier "mnist70.npz", il contient 70000 images). Attention, si vous utilisez toutes les image, votre apprentissage risque d'être fortement ralenti (cela dépend aussi de votre optimisation de code: le moins il y a de boucles, le plus vous utilisez numpy, le mieux ce sera).

5. Bonus: passez en mode semi-supervisé. Autorisez au maximum à connaitre 3 labels par cluster (idéalement, un seul). Une fois que vous avez obtenu un taux d'erreur (pour le validation set bien entendu), voyez comment la performance évolue avec les variations de l'hyper-paramètre K (nombre de clusters).

26.2 [Hors programme] Mélange de Gaussiennes et algo EM

Dans cet exercice, on suppose que vous êtes déjà familier avec l'algorithme EM.

1. Écrire le pseudo code de l'algo EM pour un modèle Gaussien: chaque *feature* est supposée suivre une loi Gaussienne (dont les paramètres dépendent de la classe) et être indépendante des autres *features*.
2. Implémenter ce code en python. Pensez bien à l'initialisation des paramètres et au critère d'arrêt.
3. Le tester sur les 2 gaussiennes de l'exercice 1.4 (4ème figure), dont le code est disponible en ligne.
4. Que pensez vous du résultat ? Est-ce impressionnant ? Quelle(s) limitation(s) voyez vous à cet exemple ?

26.3 [Hors programme] Mélange de Gaussiennes et algo EM: MNIST, apprentissage semi-supervisé

1. Tester votre algo EM (exercice 26.2) sur MNIST: pour l'ensemble d'apprentissage, on effacera les labels (qui sont pourtant disponibles, mais ici on fait un exercice). Commencer par bien énumérer les paramètres de votre modèle, vérifier que votre code est assez général, puis appliquez le aux données sans label.

2. Vous obtenez un clustering des données, mais l'ordre des clusters est aléatoire. Re-ordonnez les en "trichant" pour 1 exemple par cluster (prenez l'exemple qui est le "mieux" affecté – à vous de comprendre ce que je veux dire par là).

3. Maintenant que les clusters correspondent à un label prédit (et donc les exemples dans un cluster ont un label prédit), calculer un indicateur de performance de votre algorithme. Suivez le au cours des époques.

4. Initialement, vous avez probablement choisi un clustering en $K = 10$ clusters. Faites varier cet hyper paramètre (de 5 à 15, sans forcément essayer toutes les valeurs). Tracez la performance du modèle selon K. Que constatez vous ? Pouvait on s'y attendre ? Quelle est la limite de cette analyse ?