Fiche ML

Charles Vin

S2-2023

1 Généralité

- Fonction de perte : quantifie l'erreur associé à une décision. Erreur simple : A chaque fois qu'on se trompe, on compte 1 : 0-1 loss
- Risque : Proba de se tromper, $R(y_i|x) = \sum_j l(y_i, y_j)P(y_j|x)$ = Moyenne de la Loss pondéré par les probas à post
- Risque continue ? : $R(f) = \int_{x \in \mathcal{X}} R(f(x)|x)p(x)dx$ ($p(x) = ???$) = Esperance du X sur notre domaine continue
- iso-contours == courbe de niveau
- Une epoque = on a vu une fois tous les exemples dans le gradient
- Hinge-loss = $\max(0, 1 - yf_w(x))$
  - the Hinge loss penalizes predictions y < 1, corresponding to the notion of a margin in a support vector machine.
  - When y and $f_w(x)$ have the same sign (meaning y predicts the right class) and $|f_w(x)| \geq 1$, the hinge loss $= 0$ When they have opposite signs, the hinge loss increases linearly with $f_w(x)$ and similarly if $|f_w(x)| \geq 1$, even if it has the same sign (correct prediction, but not by enough margin).
- Lorsque les données sont de petites dimensions, le risque de sur-apprentissage est plus petit.
- Lors d'une batch de gradient, il n'est pas nécessaire de mélanger les exemples car tous les exemples sont utilisés dans chaque mise à jour de poids. VS En général, il est recommandé de mélanger les exemples lors d'une descente de gradient stochastique (SGD) afin de garantir une convergence plus rapide et une meilleure généralisation.
- $\|x\|^2 = x^T x$

1.1 Descente de gradient

Point initial x_0 et $\epsilon \geq 0$.

1. Calcul de $\nabla f(x_k)$
2. Test d'arret : si $\|\nabla f(x_k)\| \leq \epsilon$
3. Calcul d'un pas $\alpha_k > 0$ par une règle de recherche linéaire en f en x_k le long de la direction $-\nabla f(x_k)$
4. Nouvel itéré : $x_{k+1} = x_k - \alpha_k \nabla f(x_k)$

1.2 Dérivé des matrices

$$\frac{\partial(vMv)}{\partial v} = (M + M^T)V = 2MV \text{ si } M \text{ symétrique}$$
$$\frac{\partial(v^T a)}{\partial v} = \frac{\partial(a^T v)}{\partial v} = a$$
$$\frac{\partial(\log\det M)}{\partial v} = M^{-1}$$
$$\frac{\partial(Tr(AM))}{\partial v} = A$$

1.3 Multiplicateur de Lagrange

Soit $f(x)$ fonction à optimiser, sous $g(x) = 0$ contraintes d'égalités. On pose : $\mathcal{L}(x, \lambda) = f(x) - \lambda(x)$ nouvelle fonction de plus grande dimension à optimiser comme on a l'habitude en annulant le gradient : $\nabla_x \mathcal{L}(x, \lambda) = 0$. Légitiment on doit vérifier si le point est un min ou un max ou un point selle avec la matrice hessienne mais osef.

1.4 KKT & contrainte d'inégalité

Version complete de Lagrange : Fonction objectif f, g_i contrainte d'égalité, h_j contrainte d'inégalité tel que $h_i(x) \leq 0$. Fonction duale : $\mathcal{L}(x, \lambda, \mu) = f(x) - \sum_{i=1}^{p} \lambda_i g_i(x) - \sum_{j=1}^{q} \mu_j h_j(x)$. Les condition KKT pour un point x^* extremum sont

(aka résoudre le système)

$$\begin{cases} \nabla \mathcal{L}(x^*, \lambda^*, \mu^*) = 0 \\ \mu_j^* \leq 0, j = 1, \ldots, q \\ \mu_j^* h_j(x^*) = 0, j = 1, \ldots, q \end{cases}.$$

2 Arbre de décision

Algo général :

1. Déterminer la meilleure caractéristique dans l'ensemble de données d'entraînement.
2. Diviser les données d'entraînement en sous-ensembles contenant les valeurs possibles de la meilleure caractéristique.
3. Générez de manière récursive de nouveaux arbres de décision en utilisant les sous-ensembles de données créés.
4. Lorsqu'on ne peut plus classifier les données, on s'arrête.

Méthode de division des données : On vas utiliser l'entropie

Définition 2.1 (Entropie)**.** Origine de la formule de l'entropie Soit X une variable aléatoire pouvant prendre n valeurs x_i

$$H(X) = -\sum_{i=1}^{n} P(X = x_i) \log(P(X = x_i)).$$

Mesure l'homogénéité d'un dataset. C'est également la moyenne de la suprise (voir la vidéo)

Définition 2.2 (Gain d'information)**.** Mesure la réduction expects de l'entropie causé par le partitioning des exemples.

En faisant un test T sur un des attributs, on obtient deux partitions d'exemples de X : X_1 qui vérifie le test et X_2 qui ne vérifie pas le test (resp. Y_1 et Y_2).

$$H(Y|T) = \frac{|X_1|}{|X|} H(Y_1) + \frac{|X_2|}{|X|} H(Y_2).$$

Gain d'information :

$$I(T, Y) = H(Y) - H(Y|T).$$

On veut maximiser le gain d'information par le split $\Leftrightarrow$ minimiser $H(Y|T)$

3 KNN

— Prendre les K plus proche voisin pour classifier
— K petit == noisy and subject to the effects of outliers == overfitting?
— K grand == underfitting

4 Classfieur bayesien

On a :
— $P(y)$ fréquence des classe dans le dataset
— $P(x|y)$ les points de notre jeux de donnée. Graphiquement : les points coloriés

On cherche :

$$\arg\max_y P(y|x) = \arg\max_y \frac{P(x|y)P(y)}{P(x)}.$$

Naive Bayes : indépendance des dimensions de x, on peut développer le $P(x|y) = P(x_1|y) \ldots P(x_d|y)$.

Puis rapport de vraisemblance **en utilisant le risque** pour prendre la décision.

Remarque :
— Classifier bayésien = le classifier qui minimise le risque = le meilleurs classifieur possible
— Classifier optimal car minimise l'erreur car en choisissant la plus grande proba, on peut pas réduire $1 - P(y|x)$ qui est déjà le plus grand possible
— $P(x)$ difficile à calculer = répartition des points dans l'espace, dans le graph 2d non colorié. En général très petit, uniquement utile pour générer des données, pas pour faire l'argmax (aka classifier).

Autre truc important :
— On utilise classiquement une 0-1 loss
— Frontière de décision : $\frac{R(+|x)}{R(-|x)} > 1 \rightarrow$ Permet de prendre en compte les coûts asymétriques des classes. Forme dans $\mathbb{R}^2$: cercle

5 Estimation de densité

5.1 Par histogramme

Définition 5.1 (Estimation par histogramme)**.** Soit Y une v.a.r nombre de point tombant dans un bin : $Y \sim \mathcal{B}(n, p_b V)$. On a donc $E(Y) = np_bV \Leftrightarrow p_b = \frac{k}{nV}$.

— Cas discret : Comptage dans chaque classe puis normalisation par le nombre d'exemple $N \rightarrow p_b = \frac{k}{nV} = \frac{\text{Nb dans le bin}}{\text{nb d'ech tot} * \text{Volume d'un bin}}$,

— Cas continue : Discrétisation des valeurs puis comptage et normalisation

Importance de la discrétisation :

— Petit $\rightarrow$ sur-apprentissage, trou dans l'histogramme

— Trop grand $\rightarrow$ sous-apprentissage

Limite :

— Grande dimension $\rightarrow$ Perte de sens exponentiel (3 ou 4 max)

— Effet de bord : petit changement dans les bins, gros changement d'estimation.

$\rightarrow$ Solution : Estimation par noyaux

5.2 Estimation de densité par noyaux

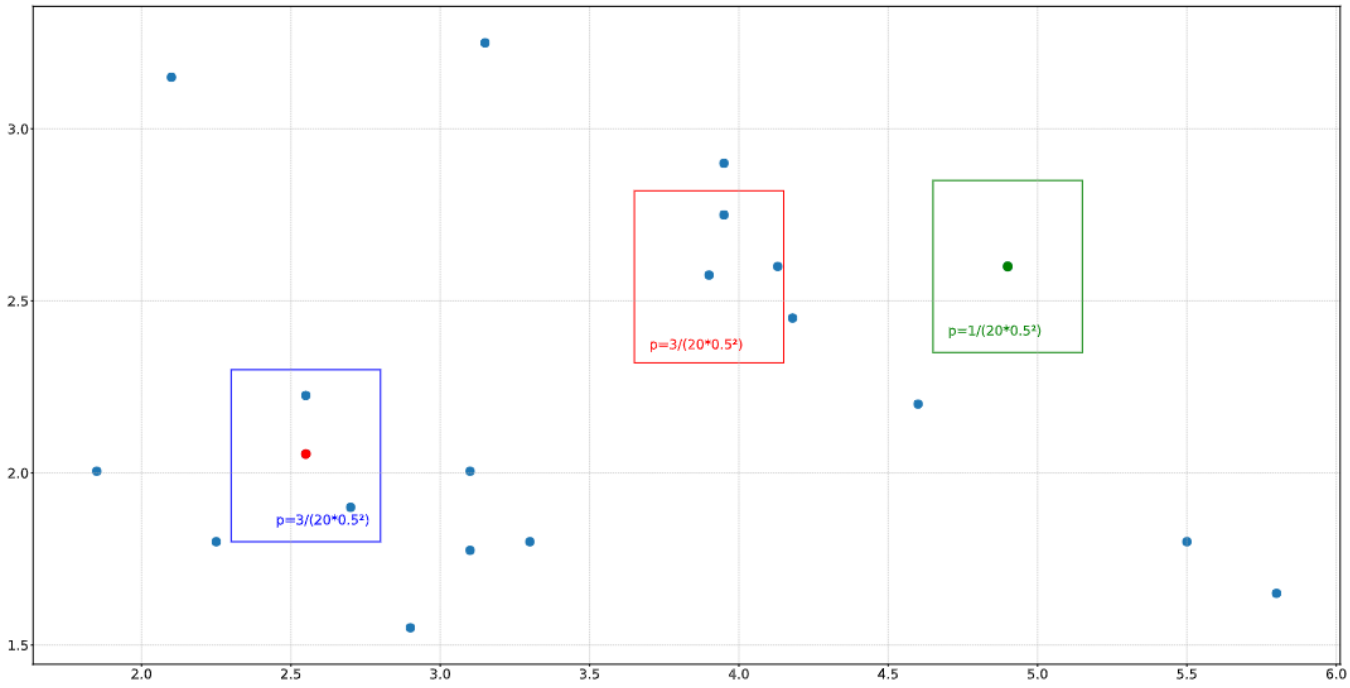

Figure 1 – Intuition de l'estimation par noyaux

Intuition figure 1 : Plutôt que de décider d'une discrétisation a priori, l'estimation est faîte en centrant une fenêtre autour du point d'intérêt x_0 (dans un espace de dimension d) à posteriori. $\rightarrow$ Problème : pas continue (si on bouge la boite et qu'un point rentre dedans, ça fait faire un saut à la fonction)

5.2.1 Fenêtre de Parzen

On combine la solution précédente avec une densité/noyaux. Classiquement Gaussien. pour obtenir un truc lisse et continue

Définition 5.2 (Fenêtre de Parzen)**.** Soit $(x_1, \ldots, x_N) \sim f$ iid

$$\hat{f}_h(x) = \frac{1}{N * h} \sum_{i=1}^{N} K(\frac{x - x_i}{h}).$$

Avec K le noyaux **centrée et réduit sur** x , souvent une fonction gaussienne. Si c'est une fonction rectangle ça fonctionne aussi. Puis y'a plein d'autre noyaux possible.

6 Regression Linéaire

— MSE : $(XW-Y)^T(XW-Y) = W^TX^TXW - (Y^TXW)^T - YXW + Y^TY$
—

$$\begin{aligned}\nabla_W MSE &= 2X^TXW - X^TY - Y^TX \\ &= 2X^TXW - X^TY - X^TY \text{ car } \lambda \in \mathbb{R}, \lambda^T = \lambda \\ &= 2X^T(XW-Y) = 0 \\ &\Leftrightarrow W = (X^TX)^{-1}X^TY\end{aligned}$$

— Sinon descente de gradient

7 Régression Logistique

— On peut pas utiliser la MSE car distance à la frontière de décision peut être très grande pour un point qui est très très très certainement dans une classe
— On vas plutot essayer de modéliser la confiance qu'on a dans la classif d'un point → Proba : $p(y=1|x) = \mu(x)$
— Modélisation de cette proba par un truc linéaire qu'on projette entre 0 et 1 avec la sigmoide ou tanh
— On remarque que le log ratio : $\log \frac{\mu(x)}{1-\mu(x)} = f_w(x)$ pour la sigmoide
— Pas de solution analytique à la log vraiss : descente de gradient

8 Perceptron

— $f_w(x) =< x, w >$, décision : $sign(f_w(x))$
— Hinge-loss = $\max(0, -yf_w(x))$, vaut 0 quand bonne prédiction
— gradient Hinge loss

$$\nabla H_w = \begin{cases} 0 & \text{si } -yxw < 0 \\ -yx & \text{sinon} \end{cases}.$$

— into descente de gradient
— le vecteur de poids w est normal à l'hyperplans de la séparatrice, $< w, x >$ mesure l'angle entre les deux vecteurs, maj : on fait bouger l'hyperpaln en fct de cette angle

Théorème de convergence : si
— $\exists R, \forall x \|x\| \leq R$
— Les données peuvent être séparées avec une marge p
— L'ensemble d'apprentissage est présenté au perceptron un nombre suffisant de fois

Alors : après au plus $\frac{R^2}{p^2}$ correction, l'algo converge

9 SVM

— Donnée non linéaire → Projection, dim ++ → Attention sur apprentissage + quel dim choisir → SMV do this auto
— Résous le problème de l'unicité de la solution également
— Maximiser la marge $\gamma \Leftrightarrow$ minimiser $\|w\|$ sous la contrainte $\forall i, (wx^i + b)y^i \geq 1$ par des calculs obscures (≥ 1 car on veut que la distance entre la droite de régression et ces deux marges soit supérieur 1)
— Prise en compte des erreurs :
  — ξ variable de débordement par rapport à sa marge pour chaque point mal classé → Raison obscure → $\xi = \max(0, 1 - (wx^i + b)y^i)$ Hinge loss
  — On avait $\min ||w||^2$ maintenant $\min ||w|||^2 + K \sum \xi$ avec K hyper param nombre d'erreur
— Optimisation avec lagrangien cas simple

$$\begin{cases} \min_{w,b} \frac{1}{2} \|w\|^2 \\ \text{s.c } y^i(wx^i + b) \geq 1 \end{cases} \Leftrightarrow L(w, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum_i \alpha_i (y^i(wx^i + b) - 1).$$

— Optimisation avec Langrangien cas complexe
—

$$\begin{cases} \min_{w,b} \frac{1}{2} \|w\|^2 + K \sum_i \xi_i \\ \text{s.c. } y^i(wx + b) \geq 1 - \xi_i, \xi_i \geq 0 \end{cases}$$

$$\Leftrightarrow \mathcal{L}(w, b, \alpha, \nu) = \frac{1}{2} \|w\|^2 + K \sum_i \xi_i - \sum_i \alpha_i (y^i(wx + b) + \xi_i - 1) - \sum_i \nu_i \xi_i$$

Kernel Tricks :

- Kernel Function : $k(x, y) =< \phi(x), \phi(y) >$
- Mesure la similarité entre 2 objets
  - – = vecteur opposé = éloigné
  - = 0 = produit orthogonal = éloigné
  - ++ = vecteur aligné = proche
- Stable pas addition, multiplication, composition avec f polynome, exponentiel
- La complexité de calcul d'un noyau polynomial est linéaire par rapport d le degré du polynome. Mais pas la dimensionnalité de la projection
-

Généralité SVM

- The support vectors are the data points that lie on the margin, which is the region between the decision boundary and the closest data points of each class. Support vectors are critical in SVM because they determine the location and orientation of the decision boundary. All other data points that are not support vectors are not used to construct the decision boundary, which means that SVM is robust to noise and outliers in the data.
- La taille de la marge est un hyper-paramètre important : marge grande == underfitting // marge petite == overfitting (séparation linéaire plus proche des points, moins centrée)

10 Réseau de neurone

10.1 Les bases

$$\frac{\partial L}{\partial w_i^h} = \sum_k \frac{\partial L}{\partial z_k^h} \frac{\partial z_k^h}{\partial w_i^h} = \sum_k \delta_k^h \frac{\partial z_k^h}{\partial w_i^h}, \text{ soit } \nabla_{\mathbf{w}^h} L = \begin{pmatrix} \frac{\partial z_1^h}{\partial w_1^h} & \frac{\partial z_2^h}{\partial w_1^h} & \cdots \\ \frac{\partial z_1^h}{\partial w_2^h} & \ddots & \\ \vdots & & \end{pmatrix} \nabla_{\mathbf{z}^h} L$$

$$\delta_j^{h-1} = \frac{\partial L}{\partial z_j^{h-1}} = \sum_k \frac{\partial L}{\partial z_k^h} \frac{\partial z_k^h}{\partial z_j^{h-1}}, \text{ soit } \nabla_{\mathbf{z}^{h-1}} L = \begin{pmatrix} \frac{\partial z_1^h}{z_1^{h-1}} & \frac{\partial z_2^h}{z_1^{h-1}} & \cdots \\ \frac{\partial z_2^h}{z_2^{h-1}} & \ddots & \cdots \\ \vdots & & \end{pmatrix} \nabla_{\mathbf{z}^h} L .$$

10.2 Les modules, leurs dérivées, leurs atoux

Paramètres :
Formule : $M^h(z^h) =$
Dérivé : $\frac{\partial M^h}{\partial w} =, \frac{\partial M}{\partial z^h} =$
Atoux :

10.2.1 Linear

Paramètre : $W \in \mathbb{R}^{input \times output}$
Formule : $M^h(z^h) = {z^h}^T * W$
Dérivé : $\frac{\partial M^h}{\partial w}(z^h) = z^h, \frac{\partial M}{\partial z^h} = W$

10.2.2 TanH

Paramètre :
Formule : $M^h(z^h) = \tanh(z^h) = \frac{\sinh z^h}{\cosh z^h} = \frac{\exp(z^h) - \exp(-z^h)}{\exp(z^h) + \exp(-z^h)} = \frac{e^{2z^h} - 1}{e^{2z^h} + 1}$
Dérivé : $\frac{\partial M}{\partial z^h} = 1 - \tanh(z^h)^2$
Atoux : Entre -1 et 1

10.2.3 Sigmoid

Paramètre :
Formule : $M^h(z^h) = \sigma(z^h) = \frac{1}{1+\exp(-z^h)}$
Dérivé : $\frac{\partial M}{\partial z^h} = \sigma(z^h) * (1 - \sigma(z^h))$
Atoux : Entre 0 et 1

10.2.4 SoftMax

Paramètres :
Formule : $M^h(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$
Dérivé : $\frac{\partial M(x_i)}{\partial x_i} = M^h(x_i) * (1 - M^h(x_i))$ plus précisement $\frac{\partial M^h(x_i)}{x_j} = \begin{cases} M^h(x_i) * (1 - M^h(x_j)) & \text{si } i = j \\ -M^h(x_j) M^h(x_i) & \text{si } i \neq j \end{cases}$
Atoux : Combo cross entropy pour plus de perf en dernière couche je crois ça se calcule mieux. Y'a une interprétation proba aussi.

10.2.5 LogSoftMax

Paramètres :
Formule : $M^h(z^h) = \log(\frac{\exp(x_i)}{\sum_j \exp(x_j)})$
Dérivé : $\frac{\partial M}{\partial z^h} =$
Atoux : When used for classifiers the log-softmax has the effect of heavily penalizing the model when it fails to predict a correct class.

10.2.6 ReLU

Paramètres :
Formule : $M^h(z^h) = \max(0, x)$
Dérivé : $\frac{\partial M}{\partial z^h} = 1$ if $x > 0$ else 0
Atoux :

10.2.7 SoftPlus/SmoothReLU

Paramètres :
Formule : $M^h(z^h) = \ln(1 + e^x)$
Dérivé : $\frac{\partial M}{\partial z^h} = \sigma(x) = \frac{1}{1+e^{-x}}$
Atoux : Approxime la relu d'une manière dérivable partout

10.2.8 LearkyReLU

Paramètres :
Formule : $M^h(z^h) = \max(\alpha x, x)$
Dérivé : $\frac{\partial M}{\partial z^h} = \begin{cases} 1 & \text{if } x > 0, \\ \alpha & \text{otherwise.} \end{cases}$
Atoux : Leaky ReLUs allow a small, positive gradient when the unit is not active.

10.2.9 MSELoss

10.2.10 Cross Entropy Loss

10.2.11 Binary CE Loss

10.2.12 CELog Soft Max

11 Non supervisé

12 Théorie de l'apprentissage

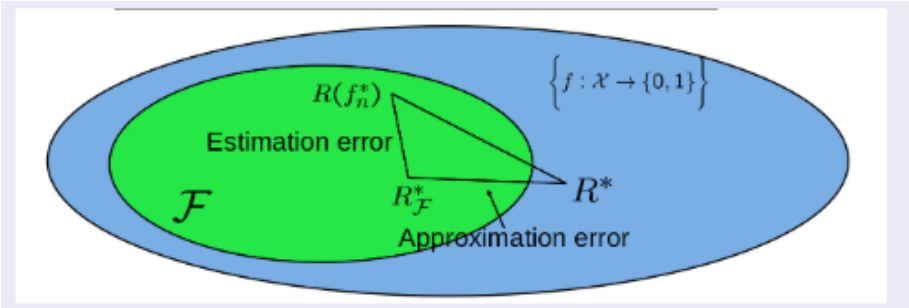

Figure 2 – Variance = estimation error, biai = approximation error

- Algorithme universellement consistant : risque de la fonction apprise converge vers le risque bayesien. Si on peut overfit sur n'importe quel jeu de donnée (comme un arbre de décision ou un réseau de neurone avec nombre de paramètre infiny).
- Biais : risque lié à la restriction de la taille de l'ensemble de fonction où on cherche la fonction optimal
- Variance : Risque lié à notre set de train et au capacité de généralisation du modèle
- Compromis biais variance : Si on augmente la taille de $\mathcal{F}$ alors on réduit le biai potentiel mais on augmente la variance potentielle
- Pour un ensemble discret $\mathcal{F}$, on peut trouver une vitesse de convergence logarithmique par rapport aux nombre de fonction $\|\mathcal{F}\|$
- Pour un ensemble infiny de fonction : de même mais utilise la VC-dimension de $\mathcal{F}$: $VC(\mathcal{F})$
- VC-dimension :
  - Un ensemble de points est shattered (pulvérisé) par un espace de fonction si pour tout partitionnement des points en deux ensembles il existe une fonction qui sépare les deux partitions.
  - La VC-dimension (Vapnik-Chervonenkis) de F sur un espace de données $\mathcal{X}$ est la taille du plus grande ensemble fini de points de $\mathcal{X}$ pulvérisé par $\mathcal{F}$.
  - Fonctions linéaires : en dimension d, VC-dimension de $d + 1$
  - le nombre d'exemples doit être linéaire en fonction de $VC(\mathcal{F})$
  - **VC-dim grande** = modèle flexible = fit/overfit sur données complexe = plus sensible au bruit // **VC-dim faible** = fonction plus "général" = moins d'overfit mais moins de performance aussi = plus robust au bruit

13 Bagging et Boosting

On veut pouvoir combiner des classfiers, ils doivent être indépendant. Comment les rendre indépendant entre eux ? Deux solutions

13.1 Bagging

- Sous échantillon du train, tirage avec remise
- Random forest = arbre de décision + bagging = randomisation du support de décision puis moyenne de tous les noeuds des arbres
- Chacun vas un peu apprendre une partie de l'espace, lié au hasard, puis les décision se moyenne
- Très efficace à grande dimensionnalité

13.2 Boosting

— Correction des erreurs faite en $n-1$ en n avec une pondération des exemples dans la loss
— Poids uniforme $\rightarrow$ On augmente le poids où il a fait des erreurs et baisse les poids des bons exemples
— $\rightarrow$ combinaison des frontière de décision
— Beaucoup d'overfit $\rightarrow$ garder des arbres de décision à faible profondeur
— Ecrit sous une forme gloutone == descente de gradient pour les arbre de décision
— AdaBoost : Si l'erreur est supérieur à 0.5 on a pas su améliorer la classif alors on arrete

14 Reinforcement Learning

14.1 Policy itération

Policy itération : policy evaluation + policy improvement, and the two are repeated iteratively until policy converges.
— $V^{\pi}(s) = E[G_t|S_t = s] = E[\sum_{k\geq 0}\gamma^k R_{t+k}]$ avec R_{t+k} les récompenss obtenues à l'état $t+k$. V^{π} représente la moyenne des récompenses possible dans le futur, futur plus où moins proche réglé par γ
— Par l'equation de Bellman

$$V_{k+1}(s) = (T^{\pi}V_k)(s) = \sum_a \pi(a|s)\sum_{s'} P(s'|s,a)[r(s,a,s') + \gamma V_k(s')].$$

— Une application répété de l'opérateur de Bellman T^{π} fait converger V_k pour évaluer la policy π
— Puis mise à jour de la politique pour chaque état en choisisant l'action qui maximise $V^k(s)$ si j'ai bien compris.

$$\pi_{k+1}(s) = argmax_a \sum_{s'} P(s'|s,a)[r(s,a,s') + \gamma V^k(s')].$$

— L'étape d'évaluation de la policy est couteuse. On peut la skip en évaluant directement chaque action pendant l'entrainement, ça fait sauter la première somme dans la formule donnant l'algo de Policy Itératio

Application :
Evaluation d'une politique π
— Initialiser π^0 avec une loi uniforme par rapport au nombre d'action disponible dans chaque état et $V_0^{\pi} = 0$ partout. Les états finaux vont rester à zéros tout le long.
— Simplifier et écrire la formule pour notre MDP, en particulier pour les états stochastiques ou non, la somme avec la proba peut disparaitre où bien se réduire.
— Tableau avec les états en colonne et les $V_k^{\pi}(s)$ en ligne, on vas faire augmenter k
— Appliquer la formule à chaque étape
Exemple d'une formule simplifiée : Voir TD9 Question 2 avec une marche aléatoire.

$$\pi(gauche|s_i)(r(s_i,g,s_{i-1}) + \gamma V_k^{\pi}(s_{i-1})) + \pi(droite|s_i)(r(s_i,d,s_{i+1}) = \gamma V_k^{\pi}(s_{i+1})).$$

Mise à jour de π : Utiliser la formule.

14.2 Value iteration

Pareil mais :
— Random value function (d'après internet)
— pour la maj de $V_k^p i$ on donne directe la valeur max. On n'évalue plus la politique mais on fait directement converger la Value fonction pour ensuite seulement choisir la politique

$$V_{k+1}(s) = \max_a \sum_{s'} P(s'|s,a)[r(s,a,s') + \gamma V_k(s')].$$

— une update final de la politique uniquement après avoir fait converger la value function. Avec la même formule qu'en policy iteration.
Atoux
— Algo plus simple
— Plus lent à converger que policy iteration.