Rapport de projet etudiant : Support Vector Machines

DEROUET Lucien

Automne 2021

Suppport Vector Machines : Étude théorique et implémentation d'un solveur

1 Introduction

Les machines a vecteurs support (ou SVM : Support Vector Machines) sont un ensemble d'algorithmes ayant une grande place dans le Machine Learning. Ils furent développés initialement par le mathématicien et informaticien russe Vapnik, dans les années 90.

Dans sa forme de base, c'est un algorithme réalisant une classification, mais il peut être étendu à résolution de problèmes de régression. Les SVM réalisent une classification binaire : un échantillon de donnée, sous forme vectorielle, est associé à une classe. Un exemple typique est la classification de mail, dont les classes peuvent être "spam" ou "non spam". Dans le cadre des SVM, les deux classes sont associées aux labels "+1" et "-1".

Les paramètres du modèles sont définis par un apprentissage dit supervisé. Pour entraîner le modèle, il faut au préalable disposer d'une quantité de données en nombre suffisant pour être représentative du problème à résoudre. Chaque échantillon de donnée est associé à un label connu, décrivant sa classe. C'est à partir de cet ensemble (nommé dataset) que l'algorithme va définir ses paramètres, de façon itérative.

Une fois l'entraînement réalisé, il est attendu que le modèle soit capable de généraliser : il prend en entrée un échantillon de données dont on ignore la classe, que le modèle va prédire. Les SVM peuvent être décrits par cette fonction : a tout échantillons de données (appartenant à l'ensemble défini par notre problème) est associé une classe.

$$f : \Re^D \to \{+1, -1\} \tag{1}$$

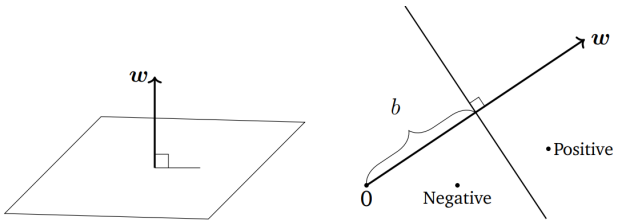
Les SVM sont une classe de modèles encore très utilisés en Machine Learning, car ils ont l'avantage d'être assez facilement interprétables. En effet, les SVM sont basés sur une intuition géométrique, ce qui rend la fonction apprise bien plus facilement visualisable que les réseaux de neurones par exemple. De plus, le problème posé par l'entraînement est un problème d'optimisation quadratique,

ce qui garanti la convergence vers le minimum absolu. Enfin, bien que les SVM soient à l'origine un classifieur linéaire, il est très facile d'y intégrer "l'astuce du noyau" (détaillée par la suite), rendant possible l'apprentissage de fonctions non linéaires. La capacité de représentation de ce type de modèle est alors démultipliée.

Contents

2 Séparation par hyperplan

L'idée derrière les SVM est assez simple : dans le cadre d'une classification binaire, il s'agit de trouver un hyper plan séparant au mieux les deux classes de données. Cet hyperplan est décrit par un vecteur qui lui est orthogonal, et un nombre b, indiquant la distance de ce plan à l'origine.



(a) Separating hyperplane in 3D

(b) Projection of the setting in (a) onto a plane

Figure 1: separating hyperplane, issu de [2]

Il est possible de trouver de quel côté du plan se trouve un échantillon de donnée x, en faisant le produit scalaire de w et x, auquel on ajoute b (pouvant être négatif). En partant du principe que w est un vecteur normé, le produit scalaire de x et w est donc égal a la longueur du projeté de x sur w. Si cette longueur est supérieure à b, x est alors de la classe positive, sinon il est de la classe négative. En prenant le label noté y, nous pouvons écrire cette règle de la façon suivante :

Dans le cas positif :

$$< w, x_n > + b \ \geq 0 \text{ pour } y_n = 1 \tag{2}$$

Dans le cas négatif :

$$< w, x_n > + b \ \leq 0 \text{ pour } y_n = -1 \tag{3}$$

Les deux formes se résument suivant :

$$y_n(< w, x_n > + b \ ) \geq 0 \tag{4}$$

3 Poser le problème d'optimisation

Afin de permettre la recherche du meilleur hyperplan, nous définissons la "marge". La marge est la distance qui sépare l'hyperplan de l'échantillon x le plus proche de celui-ci. L'idée est donc de trouver un l'hyperplan qui maximise la marge, car cela signifie que c'est l'hyperplan qui éloigne le plus les deux classes, et qui permet donc une meilleure généralisation. Une première façon de poser le problème est la suivante: nous cherchons à maximiser la marge notée r, en fonction des

paramètres w et b, tout en satisfaisant la contrainte définie précédemment, avec w un vecteur normé.

$$\begin{aligned} &\max_{w,b,r} \quad r \\ &\text{sous contraintes } y_n(< w, x_n > +b) \geq r, \|w\| = 1, r > 0 \end{aligned} \tag{5}$$

Il est possible de montrer que chercher à maximiser la marge avec un w un vecteur normal est équivalent à minimiser la norme de w, en fixant la marge à 1. Dans ce cas c'est une mise à l'échelle des données qui est réalisée, car la norme de w n'est plus fixée, et cela mène au problème d'optimisation (rendu quadratique par la mise au carré) suivant :

$$\begin{aligned} &\min_{w,b} \quad \frac{1}{2}\|w\|^2 \\ &\text{sous contraintes } y_n(< w, x_n > +b) \geq 1, \text{ pour tout } n = 1, ..., N \end{aligned} \tag{6}$$

Cette définition du problème est dite "Hard Margin SVM", car la contrainte doit être satisfaite par tous les points, elle ne permet donc pas la présence de "outliers", d'erreurs dans le dataset, au risque de compromettre l'apprentissage à cause d'un seul point, trop éloigné des autres de sa classe. Pour résoudre ce problème, et rendre plus flexible la contrainte, une "slack variable" est introduite. Nous ajoutons ξ à $y_n(< w, x_n > +b) >= 1 \ \ pour\ tout \ \ n = 1, ..., N$, pour chaque échantillon de donnée. Ainsi une contrainte n'étant pas satisfaite par un x_n sera corrigée par la valeur ξ nécessaire. Pour que ξ n'efface pas la contrainte, chaque valeur de ξ représente un coût, pondéré par l'hyper paramètre C, qu'il faut donc minimiser. Une valeur importante de C permet donc d'obtenir un hyperplan qui ne génère que très peu d'échantillons x mal classés, au dépend de la généralisation.

Quand C tend vers l'infini, le SVM soft margin tend un à être un SVM hard margin. Le Soft Margin SVM a donc la forme suivante :

$$\begin{aligned} &\min_{w,b,\xi} \quad \frac{1}{2}\|w\|^2 + C\sum_{n=1}^{N}\xi_n \\ &\text{sous contraintes } y_n(< w, x_n > +b) \geq 1 - \xi_n \\ &\qquad\qquad \xi_n \geq 0 \end{aligned} \tag{7}$$

En optimisation sous contraintes, la forme précédente est appelée la forme primale du problème, par opposition à la forme duale que nous allons introduire. La forme duale a en effet certains avantages par rapport à la forme primale. En effet, le problème dual a un nombre de paramètres égal à la taille du dataset, dont la valeur est nulle en grand majorité, et permet une résolution plus rapide que le primal.

Le problème dual repose sur la dualité lagrangienne, dans notre cas (entre autres, convexe) la dualité est dite "forte", car la solution du dual est égale à la solution du primal. Pour parvenir au dual, il faut premièrement poser le lagrangien, puis le minimiser en fonction des variables w,b et ξ (calcul du

gradient puis injecter la solution qui l'annule dans le lagrangien). La forme obtenue est le dual : en forme analytique :

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j < x_i, x_j > - \sum_{i=1}^{N} \alpha_i \\ \text{sous contraintes } & \sum_{i=1}^{N} y_i \alpha_i = 0 \\ & 0 \leq \alpha_i \leq C \text{ pour tout } i = 1, ..., N \end{aligned} \tag{8}$$

Un des grands atouts de la forme duale est qu'elle ne fait intervenir cette fois-ci un produit scalaire uniquement entre deux vecteurs x, appartenant au dataset, et non pas un produit scalaire entre le vecteur des paramètres et x, comme dans le primal. Cette spécificité permet de réaliser le "kernel trick". En effet, si nous considérons le produit scalaire comme étant la mesure de la similarité de deux vecteurs, il est possible de calculer cette similarité après transformation de ces deux vecteurs dans un autre espace (de dimension plus haute de préférence). Le produit scalaire de xi et xj est donc redéfini, après la transformation ϕ.

$$k(x_i, x_j) =< \phi(x_i), \phi(x_j) >_H \tag{9}$$

La force du kernel trick est que les fonctions kernels utilisées pour calculer la similarité ne font intervenir explicitement la transformation phi. Par exemple la fonction kernel RBF transforme les vecteurs dans un espace de dimension infinie, alors que l'expression est très simple, décrite par une exponentielle. Le développement limité de l'exponentielle nous donne une série infinie, ce qui nous donne une intuition de comment faire apparaître la transformation ϕ. Les principales fonctions kernels utilisées sont les suivantes :

$$\begin{aligned} & \text{RBF kernel : } K(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2} \\ & \text{Linear kernel : } K(x_i, x_j) = x_i^T x_j \\ & \text{Polynomial kernel : } K(x_i, x_j) = (\gamma(x_i^T x_j + 1))^d \end{aligned} \tag{10}$$

Ainsi, l'application du kernel trick permet d'augmenter grandement la capacité de représentation du SVM. Le problème de classification à résoudre est toujours linéaire, mais il est cette fois-ci réalisé après transformation des données dans un espace de dimension bien plus importante. Ainsi, le choix de la fonction kernel est crucial, puisqu'il détermine si la transformation réalisée permet un séparation linéaire des données dans l'espace d'arrivée. En pratique, le kernel RBF se montre très puissant, et est largement employé.

4 Résolution du problème

Nous allons maintenant nous intéresser à la résolution du problème dual. En appliquant le kernel trick, et en écrivant le problème dans sa forme matricielle, nous obtenons l'expression (avec e un vecteur avec uniquement des 1):

$$\begin{gathered} \min_{\alpha} \quad f(\alpha) = \frac{1}{2}\alpha^T Q\alpha - e^T\alpha \\ \text{sous contraintes } 0 \leq \alpha_i \leq C, i = 1, ..., N \\ y^T\alpha = 0 \\ \text{avec : } Q_{ij} = y_i y_j K(x_i, x_j) \end{gathered} \tag{11}$$

La résolution du problème dual en pratique pose un problème de taille : la matrice Q est la matrice rassemblant les images de la fonction kernel de tous les échantillons de données deux a deux. Le nombre d'éléments est donc égal au carré de la taille du dataset, ce qui rend impossible leur stockage en mémoire vivre. Une des approches possibles, décrite par Platt dans [4] est : SMO : Sequential Minimal Optimisation. Cela consiste à résoudre à chaque itération un sous problème à deux variables. Ce sous problème peut être résolu analytiquement. Le vecteur alpha est ensuite actualisé avec les deux nouvelles valeur, puis une nouvelle paire d'éléments est choisie.

Basé sur la méthode SMO, un des algorithmes d'entraînement les plus répandus est celui proposé dans le module LIBSVM [1]:

Result: solution α
initialization ;
$\alpha = 0$ (vecteur alpha à 0);
while *True* **do**
  if *stopping criteria is True* **then**
    exit;
  else
    i, j = working set selection ;
    solve sub problem for α_i and α_j ;
    actualize α with the solutions α_i and α_j ;
  end
end

Algorithm 1: SVM training with SMO

Nous allons étudier brièvement les 3 étapes principales de l'algorithme :

- le "Stopping criteria", qui détermine si le vecteur alpha à l'itération k est une solution du problème.
- le "working set selection" : qui consiste à sélectionner les indices i et j de alpha, afin de poser le sous problème en fonction de α_i et α_j
- la résolution analytique du sous-problème à deux variables

4.1 Stopping criteria

Le "stopping criteria" est évalué au début de chaque itération, et teste si le vecteur alpha actuel est une solution du problème, ou si il en est assez proche pour pouvoir arrêter l'entraînement. Dans un problème d'optimisation convexe sous contrainte, le minimum global est atteint lorsque les conditions KKT (Karush-Kuhn-Tucker) sont satisfaites. La première condition KKT stipule qu'à l'optimum (x^*, α_*) le gradient du lagrangien du dual est nul. Avec f_0 la fonction à optimiser et f_i la i-ème contrainte, nous avons :

$$\nabla_x L(x^*, \alpha^*) = \nabla_x f_0(x^*) + \sum_i \alpha_i^* \nabla_x f_i(x^*) = 0 \tag{12}$$

Cela peut être interprété comme le fait que les gradients de la fonction à optimiser se retrouve colinéaire et opposé au gradient des contraintes. Ainsi, il n'y a plus de progrès possible. La deuxième condition KKT stipule que le produit de la contrainte en x_i et le multiplicateur de lagrange α_i correspondant doit être nul, pour tout i. Nous pouvons en déduire que α_i est à 0 quand la contrainte n'est pas active, et α_i a une valeur non nulle quand la contrainte est active. Dans ce dernier cas, l'échantillon x associé est un vecteur support, sur lequel est posée la marge. En posant les conditions KKT à partir du dual, nous obtenons la forme suivante, avec λ et ξ des vecteurs non nuls :

$$\begin{gathered} \nabla f(\alpha) + by = \lambda - \xi \\ \lambda_i \alpha_i = 0; \quad \xi_i(C - \alpha_i) = 0; \quad \lambda_i \geq 0; \quad \xi_i \geq 0, \quad i = 1, ..., N \end{gathered} \tag{13}$$

Les conditions permettent d'obtenir la forme

$$\begin{cases} \nabla_i f(\alpha) + by_i \geq 0 \text{ if } \alpha_i < C \\ \nabla_i f(\alpha) + by_i \leq 0 \text{ if } \alpha_i > 0 \end{cases} \tag{14}$$

En multipliant des deux côtés par y_i (étant égal à 1 ou -1), il est possible de réécrire la condition sous la forme :

$$m(\alpha) \leq b \leq M(\alpha)$$

Avec

$$m(\alpha) = \max_{i \in I_{up}(\alpha)} -y_i \nabla_i f(\alpha) \quad \text{et} \quad M(\alpha) = \min_{i \in I_{low}(\alpha)} -y_i \nabla_i f(\alpha) \tag{15}$$

et

$$I_{up}(\alpha) = \{t | \alpha_t < C, y_t = 1 \text{ ou } \alpha_t > 0, y_t = -1\} ,$$
$$I_{low}(\alpha) = \{t | \alpha_t < C, y_t = -1 \text{ ou } \alpha_t > 0, y_t = 1\}$$

Ce résultat peut être retrouvé assez facilement en posant bien tous les cas possibles. Cette forme est intéressante, car elle permet de déduire à partir des nombres m(alpha) et M(alpha) si les conditions KKT sont respectées ou non. Si la condition est violée, donc m(alpha) ¿ M(alpha), l'entraînement doit être poursuivi. Dans l'implémentation, l'entraînement sera stoppé si le m et M se trouvent suffisamment proches, à 10^-8 près par exemple.

4.2 Working set selection

Une fois qu'il est assuré que l'entraînement doit être poursuivi (condition KKT violée, donc $m(\alpha) > M(\alpha)$), il faut choisir les indices i et j pour construire le sous-problème aux deux variables α_i et α_j.

Ce problème n'est pas simple, plusieurs solutions sont décrites dans [3], dont celle utilisée par LIBSVM. Le point de départ est la condition KKT violée, donc $m(\alpha) > M(\alpha)$. Nous pouvons imaginer qu'il soit possible de violer "plus ou moins" la condition, en fonction de l'écart entre $m(\alpha)$ et $M(\alpha)$, car si m et M sont très proche, la condition KKT peut être violée, mais de très peu. Ainsi, en prenant (avec α^k le vecteur alpha à l'itération k)

$$\begin{aligned} i \in & \arg \max_t \{-y_t \nabla f(\alpha^k)_t | t \in I_{up}(\alpha^k)\} \\ j \in & \arg \min_t \{-y_t \nabla f(\alpha^k)_t | t \in I_{low}(\alpha^k)\} \end{aligned} \tag{16}$$

Nous obtenons les indices i et j correspondant aux α_i et α_j violant le **plus** la condition KKT, parmi les indices qui violaient déjà cette condition (car i appartient a I_{up} et j appartient à I_{low})

Ce working set est acceptable, mais peut être grandement amélioré. En effet, la publication part de l'intuition qu'il existe un lien entre la "maximum violating pair" (donc la paire i et j, trouvée précédemment) et l'approximation du premier ordre de la fonction à optimiser.

En effet, soit $d^T = [d_B^T, 0_N^T]$ avec B = i,j et N la dimension du vecteur alpha - 2, et $-1 <= d_B <= 1$, nous pouvons poser à l'itération k : $f(\alpha^k + d)$, l'état de la fonction à l'itération suivante. Comme nous cherchons à minimiser cette fonction, cela revient à minimiser $f(\alpha^k + d) - f(\alpha^k)$

Or cette forme peut être écrite comme étant l'approximation du premier ordre de f, donc on cherche à minimiser : $f(\alpha^k + d) - f(\alpha^k) = \nabla f(\alpha^k)_B^T d_B$

Les auteurs trouvent alors que, de façon surprenante, i et j (les mêmes que trouvés précédemment) sont les argument correspondant au minium (arg min) de l'expression précédente.

Ainsi, les auteurs développent une nouvelle méthode, cette fois-ci basée sur l'approximation de deuxième ordre, construite de la sorte :

$$\begin{aligned} f(\alpha^k + d) - f(\alpha^k) &= \nabla f(\alpha^k)^T d + \frac{1}{2} d^T \nabla^2 f(\alpha^k) d \\ &= \nabla f(\alpha^k)_B^T d_B + \frac{1}{2} d_B^T \nabla^2 f(\alpha^k)_{BB} d_B \end{aligned} \tag{17}$$

Sans rentrer dans plus de détails (la publication développe le raisonnement complet), le "working set selection" consiste à trouver i comme précédemment, puis à trouver le minimum suivant j, de l'approximation du second ordre, i étant connu.

Pour tout t,s on pose

$$a_{ts} = K_{tt} + K_{ss} - 2K_{ts} \text{ et } b_{ts} = -y_t \nabla_t f(\alpha^k) + y_s \nabla_s f(\alpha^k) > 0$$

Puis, poser i et j tels que

$$\begin{aligned} i &\in \arg\max_t \{-y_t \nabla_t f(\alpha^k) | t \in I_{up}(\alpha^k)\} \\ j &\in \arg\min_t \{\frac{-b_{it}^2}{a_{it}} | t \in I_{low}(\alpha^k), -y_t \nabla_t f(\alpha^k) < -y_i \nabla_i f(\alpha^k)\} \\ &\text{avec } K_{ij} = K(x_i, x_j) \end{aligned} \tag{18}$$

4.3 Résolution analytique

Une fois les indices i et j connus, le problème est réduit aux deux variables α_i et α_j. Bien que la résolution semble être simple telle que présentée dans la publication Libsvm, une description plus détaillée mais assez évasive est donnée dans la publication de Platt. Enfin, la meilleure description a été trouvée sur le site de l'université de tel aviv, sur cet article

(la publication n'a pas d'auteur), à la page 9 les calculs sont posés, et semblent très complexes. Ces calculs n'ont donc pas étés refaits contrairement aux autres présentés. Heureusement, l'implémentation de la résolution est simple, un exemple de code est fourni avec la publication libsvm, et a été reproduit à l'identique dans mon l'implémentation.

4.4 Implémentation réalisée

Une fois l'étude théorique réalisée, l'implémentation d'un solveur de SVM a été faite sous python, en n'utilisant que le package Numpy pour la manipulation des vecteurs et matrices. Le "stopping criteria", le "working set selection", les calculs de kernel, de gradients, la prédiction et boucle d'entraînement ont été implémentés uniquement à partir des formules. Seule la résolution analytique du sous-problème a été recopiée à partir de la publication [1] L'implémentation réalisée propose un entraînement sur quelques datasets à deux dimensions, générés aléatoirement, ainsi que la visualisation de la fonction apprise par le modèle. Des images sont disponibles en annexe

Même si l'implémentation privilégie l'usage de fonctions optimisées de numpy, notamment numpy.dot, employant la bibliothèque BLAS, les performances de calculs peuvent être améliorées avec l'usage d'un cache. En effet, le calcul de quelques colonnes de la matrices de gram est une des étapes les plus lourdes réalisées par itération. Un système de cache permettrait de garder en mémoire les colonnes récemment utilisées, et ainsi éviter de les calculer des nouveau. Enfin, la publication Libsvm propose un système de "shrinking" pour réduire la taille de problème, mais il n'a pas été étudié.

Le code est disponible sur mon git

5 Conclusion

Pour conclure, ce projet a été l'occasion d'étudier en profondeur un des algorithmes central du machine learning. J'ai été surpris par la richesse mathématique autour de la façon de poser le problème, mais surtout de le résoudre. Cela a été d'autant plus difficile que la documentation se concentre en majorité sur la construction du problème, sans s'intéresser aux méthodes de résolutions.

Ce projet m'a aussi permis de progresser en algèbre, et de découvrir l'optimisation sous contrainte.

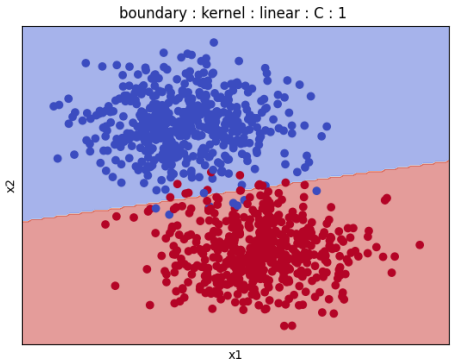
References

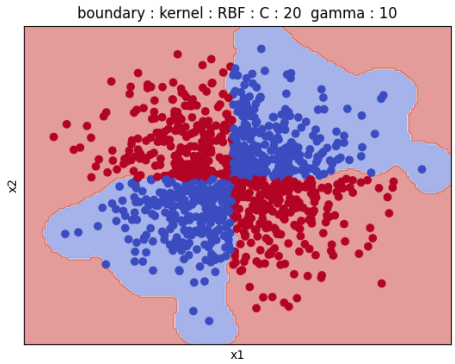
[1] Chih-Chung Chang and Chih-Jen Lin. "LIBSVM: A library for support vector machines". In: *ACM Transactions on Intelligent Systems and Technology* 2 (July 2007).

[2] Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong. *Mathematics for Machine Learning*. Cambridge University Press, 2020.

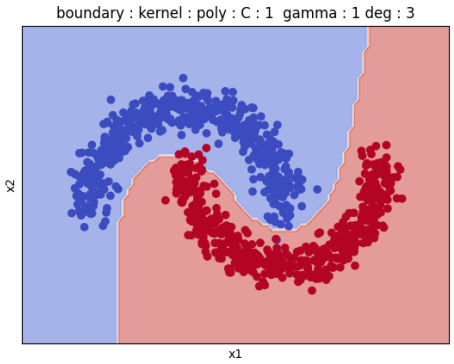
[3] RE Fan, PH Chen, and Chih-Jen Lin. "Working Set Selection Using Second Order Information for Training SVM". In: *Journal of Machine Learning Research* 6 (Jan. 2005), pp. 1889–1918.

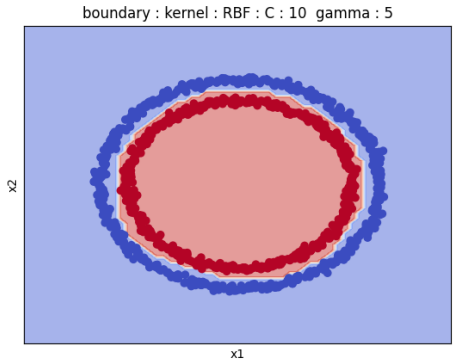
[4] John Platt. "Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines". In: *Advances in Kernel Methods-Support Vector Learning* 208 (July 1998).





(a) accuracy on test 0.976, accuracy on train 0.968 (b) accuracy on test 0.983, accuracy on train 0.998





(c) accuracy on test 0.992, accuracy on train 0.990 (d) accuracy on test 1, accuracy on train 1

6 Annexe

Les quatre dataset générés sont les suivants