MASTER EN SCIENCES INFORMATIQUES

Machine learning
gestion d'algorithmes d'apprentissage supervisés

Mignolet, Adrien

Award date:
2016

Awarding institution:
Universite de Namur

Université de Namur
Faculté d'informatique
Année académique 2015–2016

Machine learning

Gestion d'algorithmes d'apprentissage supervisé

Adrien Mignolet

Promoteur : _____ (Signature pour approbation du dépôt - REE art. 40)
Benoît Frénay

Co-promoteur : Renaud Lambiotte

Mémoire présenté en vue de l'obtention du grade de
Master en Sciences Informatiques.

Résumé

Le présent mémoire propose une description des principes inhérents aux d'algorithmes décisionnels issus de l'apprentissage automatique (*machine learning*). L'apprentissage automatique est l'une des disciplines associées à l'appellation très générique « Big data » qui désigne l'exploitation de volumes de données à ce point importants que les méthodes traditionnelles s'avèrent inefficaces et inadaptées.

L'étude explore plus particulièrement les algorithmes d'apprentissage supervisé, une classe d'algorithmes qui s'améliorent sur base d'exemples dont le résultat est connu.

Chacune des méthodes étudiées fera l'objet d'une application pratique pour illustrer sa construction et son implémentation en y évoquant les bonnes pratiques en la matière. C'est ainsi, par exemple, que les algorithmes de classification seront mis à l'épreuve dans un cas pratique de reconnaissance de caractères.

Mots-clés : Apprentissage automatique, Big data, Apprentissage supervisé, Régression, Classification, Machine à vecteurs de support, Réseau neuronal, Scikit-Learn

Summary

This master's dissertation provides a description of some principles inherent in the implementation of decision-making algorithms according to machine learning theories. Machine learning is one of the disciplines that belongs to the very generic term "Big data" which refers to the exploitation of data volumes so huge that traditional methods are ineffective and inadequate.

This document focuses on supervised learning algorithms, a class of algorithms that improve with data in which the result is known.

Each method studied will be applied practically to illustrate its construction and implementation discussing, by the way, about good practices. For example the classification algorithms will be finally tested in a practical case of character recognition.

Keywords : Machine Learning, Big data, Supervised learning, Regression, Classification, Support vector machine, Neural network, Scikit-Learn

Ce mémoire est le résultat de près d'une année de documentation et d'apprentissage sur le *Machine learning*. Au-delà des ressources documentaires utilisées, sa réalisation a naturellement été rendue possible grâce au concours de plusieurs personnes qui y ont contribué de diverses manières.

Je tiens, d'abord, à remercier Monsieur Frénay, Professeur à l'Université de Namur, qui a accepté de prendre en charge la direction de ce mémoire en qualité de promoteur ainsi que Monsieur Lambiotte, Professeur à l'université de Namur lui aussi, qui en a assuré la co-promotion.

Je tiens également à exprimer toute ma gratitude à Giuseppe Albanese, l'un de mes collègues de la société NRB qui a été l'auteur de nombreuses suggestions de lectures documentaires.

Ce travail est enfin l'aboutissement de mon cursus à horaire décalé, une épreuve de taille pour qui s'est lancé dans la vie professionnelle depuis plusieurs années.

Merci du fond du cœur à mon épouse, mes deux enfants et à mes parents pour leur soutien indéfectible, leur compréhension et leur confiance.

Table des matières

I. Introduction

En 2003, un centre hospitalier de New York[KRIS 2013] spécialisé en cancérologie a recruté un spécialiste en diagnostic médical réputé pour ses capacités hors du commun. Il s'appelle Watson et a étudié plus de deux millions de pages issues de revues médicales, d'articles et de rapports de patients. Watson est un boulimique de connaissances et tous les domaines l'intéressent. C'est ainsi qu'il a même remporté le jeu télévisé, bien connu au Etats-Unis, « Jeopardy! ».

Voilà un profil bien particulier que les hôpitaux et centres de recherche les plus prestigieux auraient probablement aimé acquérir en leur sein… On pourrait penser que le *Memorial Sloan Kettering* Cancer Center a engagé d'âpres négociations pour décrocher sa signature avant tout autre établissement. Et pourtant il n'en est rien… Watson possède également la faculté de travailler pour plusieurs hôpitaux en même temps ! Watson est maintenant à la disposition de réels patients, même issus d'hôpitaux différents…

Il ne s'agit naturellement pas d'une personne réelle ayant étudié la médecine mais d'un algorithme d'intelligence artificielle réalisé par la société IBM. Il incarne l'un des meilleurs exemples de produit d'une discipline issue de l'informatique que nous appelons le « *Machine Learning* », connue en français comme l'apprentissage automatique ou artificiel. Il repose sur le principe que beaucoup d'activités ont été observées et mesurées et que ces informations pourraient nous aider à ajuster nos décisions futures, voire à mettre en évidence des choses que nous ignorions jusqu'alors.

[KRIS 2013] Kris Mark G., *Memorial Sloan Kettering's Mark Kris on Partnership with IBM Watson*, https://www.mskcc.org/blog/msk-mark-kris-partnership-ibm-watson, 8 février 2013, consulté le 10 janvier 2016

« L'utilisation conjointe de quantités massives d'informations et d'algorithmes d'apprentissage relativement simples rend possible la solution de problèmes considérés il y a peu comme inaccessibles. »[LEMBERGER 2014]

Pirmin Lemberger, Data scientist de la société Weave et auteur d'une revue critique sur les technologies informatiques.

Cette discipline est elle-même une composante du domaine du « *Big data* », un concept dont les frontières sont floues et dont il n'existe pas réellement de définition. D'abord utilisé comme un « *base word* », le « *Big data* » désigne le fait que la disponibilité de nombreuses données couplée à l'utilisation d'un algorithme générique entrainé avec celles-ci s'avère plus convaincant et efficace que la réalisation d'un logiciel décisionnel sans ces données… Le concept de « *Big data* » est cependant plus large. Dans celui-ci, on retrouve également les problématiques d'acquisition, de transformation et de stockage de ces données, l'apprentissage automatique se concentrant pour sa part sur leur exploitation.

« Big data is not about the data. »[1]

Gary King, Professeur à l'université d'Harvard

Si l'existence et la pertinence des données est un préalable, c'est l'analyse et l'affinement d'un algorithme d'apprentissage qui constitue la valeur ajoutée majeure. Il s'agit de l'activité critique pour la réussite de tout objectif de conception d'algorithme puissant et précis.

Au-delà de la difficulté de collecter et de stocker l'information, c'est donc la nécessité d'utiliser des algorithmes efficaces qui représente l'une des plus grandes difficultés. Bien que de nombreuses librairies soient maintenant accessibles et utilisables librement, seule la compréhension de leur fonctionnement permet de faire un choix opportun face à un besoin

[LEMBERGER 2014] Lemberger Pirmin, *Le « machine learning » – quand les données remplacent les algorithmes*, http://www.journaldunet.com/solutions/expert/56923/le---machine-learning-----quand-les-donnees-remplacent-les-algorithmes.shtml, 28 mars 2014, consulté le 10 janvier 2016

[1] « Big data is not about the data. » est le nom d'une conférence donnée par Gary King, Professeur à l'université de Harvard, qu'il a notamment tenue au « Golden Seeds Innovation Summit » à New York le 30 janvier 2013 où il défend l'idée que la valeur ajoutée du Big data repose essentiellement sur l'analyse et les algorithmes traitant les données et non les données elles-mêmes.

identifié. Des applications plus complexes requièrent d'ailleurs la combinaison de plusieurs techniques.

Les applications sont nombreuses ! La collecte des données est aujourd'hui généralisée sur l'essentiel des services liés à l'informatique. Toutes transactions bancaires, toutes connexions ou interactions sur le contenu d'un service informatique (comme un site web) laisse aujourd'hui une trace dans une base de données ou un fichier journal. En parallèle, tous les services historiquement détachés de l'informatique ont intégré ou intègrent à présent les moyens informatiques de leurs processus, des services postaux aux écoles et administrations. Ces données s'accumulent, parfois sans même qu'aucun moyen spécifique n'ait été développé pour leur exploitation.

« Cinq exaoctets d'informations ont été créés depuis l'aube de l'humanité à l'année 2003. Ce volume d'informations est aujourd'hui créé en deux jours. »[2]

Eric Schmidt, ancien PDG de Google, aujourd'hui président du conseil d'administration

On considère, aujourd'hui, que la quantité d'informations conservées double tous les 18 mois[3].

Longtemps réservées aux statistiques, à la sécurité ou aux exigences de traçabilité, ces données représentent aujourd'hui une mine d'informations sur leurs domaines respectifs.

C'est ainsi que nombreux auteurs, comme Antoinette Rouvroy dans sa publication « Of data and me » Fundamental Rights and Freedoms in a World of Big Data[ROUVROY 2016] s'inquiètent, par exemple, des conclusions que pourrait tirer tout organisme pouvant recouper celles-ci. Par l'accès à de nombreuses données personnelles, le risque de pouvoir identifier un individu est énorme même si ces données sont rendues anonymes au préalable. L'étude[DE MONTJOYE 2015] « Unique in the shopping mall: On the re-identifiability of credit card metadata » menée par de

[2] Déclaration publique d'Eric Schmidt lors du « Guardian Activate 2010 summit » le 1er juillet 2010

[3] Pour l'anecdote, cette affirmation est très régulièrement attribuée par erreur à Gordon Earle Moore, l'auteur de la loi de Moore spécifiant que la quantité de transistors des microprocesseurs tendait à doubler tous les deux ans. Son origine est pourtant inconnue.

[ROUVROY 2016] Rouvroy Antoinette, *« Of Data and Men »*, *Fundamental Rights and Freedoms in a World of Big Data*. Council of Europe, Directorate General of Human Rights and Rule of Law, 2016)

[DE MONTJOYE 2015] de Montjoye Yves-Alexandre, « Unique in the shopping mall: On the re-identifiability of credit card metadata », Science 347, 30 janvier 2015

Yves-Alexandre de Montjoye, chercheur dans le domaine du respect de la vie privée dans le monde numérique à l'Université de Harvard, démontre, par exemple, la faisabilité de cette ré-identification à hauteur de 90% des détenteurs de carte de banque sur base de leur numéro de compte et l'historique des transactions sur trois mois pour 1,1 millions d'utilisateurs sans connaitre au préalable leur nom ou adresse.

L'exploitation des données représente l'un des enjeux importants pour les acteurs participant à la mise au point de services dits intelligents. Les observateurs critiques des tendances technologiques en parlent, d'ailleurs, comme l'un des défis majeurs de cette décennie.

« Cachée dans l'amoncellement des données existe la connaissance qui pourrait changer la vie d'un patient, ou changer le monde. »

Atul Butte, Professeur de la faculté de médecine de l'université de Stanford

Fil conducteur de ce mémoire

Par le biais de différents exemples et de l'explication des principes mathématiques qui s'y cachent, le présent mémoire tend à faire un tour d'horizon des méthodes en avançant graduellement dans leur complexité afin de démonter l'intérêt de celles-ci dans la résolution de problèmes complexes.

La première partie de ce mémoire débutera par une explication générale de la modélisation d'une hypothèse et de sa minimisation.

Ce mémoire traitera ensuite de la qualité d'une représentation mathématique et des indicateurs de performance qui y sont liés. Le vocabulaire spécifique caractérisant les modèles y sera défini au même titre que les bonnes pratiques sur la construction de ceux-ci par une démarche d'entrainement.

La seconde partie s'entame avec un premier exemple simple basé sur une régression linéaire. Celui-ci illustre en pratique les étapes clés de réalisation d'un modèle mathématique et de son entrainement.

La question de la régularisation sera, ensuite, étudiée sur un second exemple de régression non-linéaire et poursuivie par une généralisation à une régression non-linéaire dont les données

d'entrées comportent plusieurs caractéristiques. Les concepts de régularisation et de normalisation seront étudiés pour rendre ces algorithmes performants.

La présentation d'autres familles d'algorithme utiles à la classification, tels que les régressions logistiques, les réseaux neuronaux, les machines à vecteur de support (*Support Vector Machine*), viendra compléter ce chapitre. Par souci de concision, seuls certains concepts du *Machine Learning* nécessaires à une bonne entrée en matière sont étudiés. Le présent mémoire n'a donc pas la prétention d'être exhaustif.

Dans la même idée, les autres domaines associés au « *Big data* » tels que les problématiques du stockage, de l'acquisition de données (*Data Mining*) ne seront pas abordés.

Une partie pratique suit, enfin, la description théorique des fondements du *Machine Learning*. Celle-ci contient un exemple concret de classification de caractères manuscrits pour illustrer et comparer ces méthodes entre elles.

Les scripts écrits à titre d'exemple sont repris dans leur intégralité en annexe. Il ne s'agit pas d'exemples minimaux mais de ceux qui ont servi à illustrer cette publication. Ils peuvent être utiles au lecteur qui souhaiterait se familiariser avec ces concepts par leur mise en pratique.

Objectif

L'objectif du présent document est de démontrer l'intérêt du *Machine Learning* dans la réalisation d'algorithme décisionnel et dans l'étude de données complexes. Il tente de familiariser son lecteur à ses principes, ses contraintes et son vocabulaire. Ce mémoire a l'ambition d'aider à la sélection des méthodes au regard d'un besoin identifié. Il apporte également une explication vulgarisée de ces méthodes pour expliquer leur comportement.

Choix technologiques

Tels que me l'ont suggérés mes promoteurs, l'ensemble des programmes ont été écrits en Python au moyen des outils SciPy[4] et Scikit-Learn[5]. Ces outils rassemblent, en effet, toutes les bibliothèques nécessaires au traitement mathématique de données de manière optimale. Ces

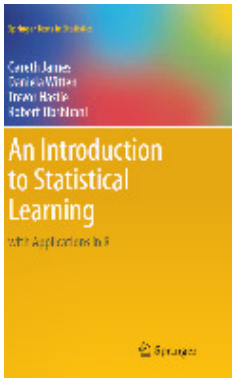
[4] https://www.scipy.org/
[5] http://scikit-learn.org/

suites sont maintenues par une vaste communauté active dans le domaine du *Machine Learning*. Elles possèdent l'avantage d'être libres, performantes, bien documentées et gratuites.

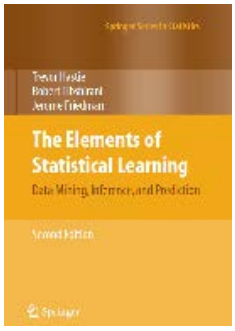
Sources

De nombreuses sources documentaires ont été utilisées pour la rédaction du présent document. Certains ouvrages de référence ont cependant influencé plus lourdement le choix des sujets sélectionnés. Citons notamment :

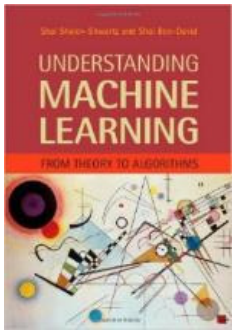
- le cours en ligne « Introduction to Machine Learning » de l'Université de Stanford dispensé par Andrew Ng et accessible librement sur la plateforme Coursera[6] ;
- le livre « An introduction to Statistical Learning » de Gareth James, Daniela Witten, Trevor Hastie et Robert Tibshirani ;



- le livre « The Elements of Statistical Learning » de Trevor Hastie, Robert Tibshirani et Jerome Friedman ;



- le livre « Understanding Machine Learning : From Theory to Algorithms » de Shai Shalev-Shwartz et Shai Ben-David



[6] https://www.coursera.org/

II. Principes généraux

Les algorithmes d'apprentissage automatique (*Machine learning*)[NILSSON 1998] ont pour principe de lire un grand nombre de données collectées ou mesurées au préalable pour déterminer leurs décisions futures, quel qu'en soit l'objet. Cela passe par la définition d'une représentation mathématique déduite de l'analyse des données d'entrée qui prend la forme d'une fonction mathématique, souvent complexe, qui sera le résultat de la phase d'apprentissage.

Cette représentation pourrait, par exemple :

- prédire une valeur numérique au regard de paramètres d'entrée (par le recours à une régression linéaire ou non linéaire) ;
- déterminer la probabilité d'appartenance à des ensembles identifiés (grâce aux techniques de classification, de régression logistique) ;
- mettre en évidence des groupes homogènes de données…

La capacité à apprendre d'un algorithme permet de s'adapter au changement du domaine d'application. La collecte de nouvelles informations peut à tout moment compléter les données initiales pour entrainer à nouveau l'hypothèse et parfaire la prise de décision.

Dans la suite du mémoire, nous étudierons les moyens à mettre en œuvre pour réaliser cette représentation et quelles sont ses utilisations utiles.

A. Fonction d'hypothèse

Comme évoqué précédemment, la définition de la représentation mathématique est généralement réalisée par approchements successifs de la fonction idéale. Cette fonction est ajustée par la modification de paramètres en plusieurs étapes pour réduire le taux d'erreur. Une fonction naïve initialisée, puis, améliorée lors d'une phase dite d'entrainement.

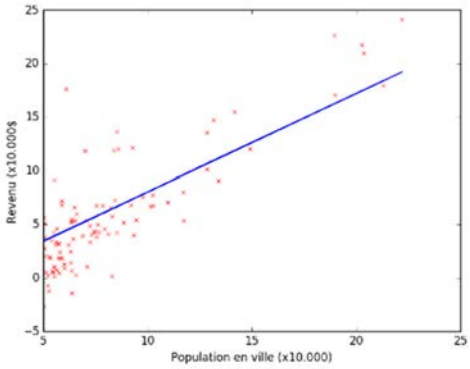
[NILSSON 1998] Nilsson Nils J., *Introduction to Machine Learning,* Stanford University : Department of Computer Science, 1998, 1-3 p. (*Introduction*)

Cette méthode présente l'avantage de fonctionner dans davantage de cas que lorsque cette fonction est calculable directement (une régression linéaire peut, par exemple, être calculée par la méthode des moindres carrés).

La phase d'apprentissage s'initialise au moyen d'une fonction d'hypothèse notée : $h_\theta(x)$. Celle-ci peut prendre différentes formes suivant le type d'apprentissage envisagé et la forme que celle-ci devrait épouser.

$$h_\theta(x) = \theta_0 + \theta_1 x$$

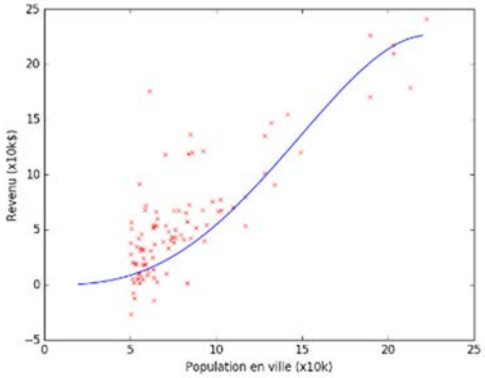
(régression linéaire)



Régression linéaire, 1. [données remises par l'université de Stanford à des fins pédagogiques sur Coursera.org (Introduction to Machine Learning), 2015]

$$h_\theta(x) = \theta^T x = \theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_n x^n$$

(régression non linéaire polynomiale)



Régression non linéaire, 1. [données remises par l'université de Stanford à des fins pédagogiques sur Coursera.org (Introduction to Machine Learning), 2015]

$$h_\theta(x) = \theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_2 + \theta_4 \log x_2 + \theta_5 x_1 x_2 + \cdots + \theta_o x_n$$

(régression non linéaire plus complexe)

Où θ représente le vecteur contenant le poids de chacune des variables (*features*). x représente le jeu de données d'entrée ; θ^T représente la transposée[7] du vecteur θ

Dans la plupart des méthodes que nous étudierons plus loin, la fonction d'hypothèse est déterminée selon une forme spécifique et dotée de variables fixées arbitrairement, il est donc peu probable que celle-ci puisse épouser, même vaguement, les données d'entrées avant tout ajustement. L'idée est, en effet, de la modifier successivement afin de déterminer une fonction généralisant de manière satisfaisante (mais pas parfaite) les données d'entrées tel que cela sera détaillé dans les sections suivantes.

B. Evaluation de l'hypothèse et ajustement des variables

La qualité de l'évaluation de la fonction d'hypothèse s'effectue en mesurant son éloignement avec les données d'entrée[JAMES 2015]. C'est ce que nous appelons le coût.

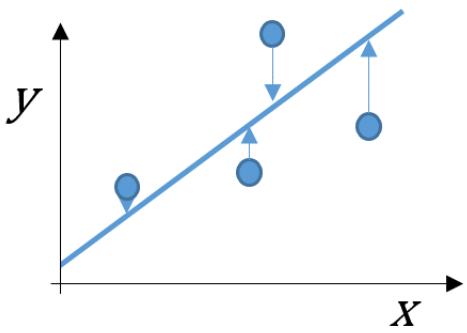
Celui-ci est calculé par la fonction de coût (*Cost function*) qui réalise la somme des différences entre les données observées et celles calculées par la fonction d'hypothèse et par celles que nous possédons dans notre jeu de données. Plus celui-ci est élevé, plus l'hypothèse est éloignée des données d'entrée.

Nous noterons cette fonction $J(\theta_0, \theta_1, \dots, \theta_n)$ où $\theta_0, \theta_1, \dots, \theta_n$ représentent les variables de notre fonction hypothèse (n exprime donc le nombre de variables présentes dans la fonction d'hypothèse).

Le coût peut, par exemple, être mesuré par le rapport entre le nombre de prévisions incorrectes sur le nombre total de prévisions ou encore par le calcul du carré de la différence entre les valeurs calculées par la fonction hypothèse et les données observées.

[7] La transposée correspond à l'inversion des lignes et colonnes d'une matrice. Concrètement, la première ligne devient la première colonne et ainsi de suite.

[JAMES 2015] James G., Witten D., Hastie T., Tibshirani R., *An Introduction to Statistical Learning*, 2e édition, New York : Springer, 2015, 15-18 p. (2.1 What Is Statistical Learning?)



Mesure du coût par projection verticale [illustration inspirée d'un article de Weisstein, Eric W. (http://mathworld.wolfram.com/LeastSquaresFitting.html)]

La fonction généralement utilisée prend la forme suivante :

$$J(\theta_0, \theta_1, \dots, \theta_n) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

où $y^{(i)}$ représente le résultat observé de la donnée dont l'indice est i du jeu de données d'entrée ; $h_\theta(x^{(i)})$ exprime l'évaluation de la même donnée évaluée grâce à la fonction hypothèse. Le facteur $\frac{1}{2m}$ veille à limiter la croissance du coût lorsque la quantité de données d'entrée augmente.

Le coût se calcule cependant parfois différemment suivant le type d'apprentissage. Sa fonction diffère, par exemple, pour une régression linéaire et une régression logistique. Cette fonction s'avère utile lors de l'entrainement mais également a posteriori pour mesurer l'efficacité de l'apprentissage.

Fonction de coût adaptée à une régression linéaire ou non linéaire :

$$J_{test}(\theta) = \frac{1}{2m_{test}}\sum_{i=1}^{m_{test}}(h_\theta\left(x_{test}^{(i)}\right) - y_{test}^{(i)})^2$$

Fonction de coût adaptée à une régression logistique :

$$J_{test}(\theta) = -\frac{1}{m_{test}}\sum_{i=1}^{m_{test}} y_{test}^{(i)} \log h_\theta\left(x_{test}^{(i)}\right) + \left(1 - y_{test}^{(i)}\right)\log h_\theta(x_{test}^{(i)})$$

Cette fonction renseignant un indicateur du taux d'erreur, celle-ci est parfois simplifiée en régression logistique, par le simple calcul du taux de classification erronée (*misclassification error* ou *accuracy*)

$$J_{test}(\theta) = \frac{\text{Nombre d'erreurs}}{\text{Nombre de prédictions}}$$

C. Optimisation de l'hypothèse

Différentes techniques, dites d'optimisation, permettent de minimiser le coût de notre hypothèse en ajustant les variables d'entrée notées $\theta_0, \theta_1, \dots, \theta_n$.

Citons notamment :

- le gradient descendant, également appelé l'algorithme du gradient ;
- le gradient descendant stochastique ;
- le gradient descendant mini-batch ;
- les algorithmes BFGS et L-BFGS;
- le gradient conjugué.

Toutes ces méthodes guident la phase d'entrainement avec l'objectif de réduire le taux d'erreur.

1. Le gradient descendant (Gradient Descent)

Afin d'optimiser la fonction d'hypothèse, les variables qui la composent sont adaptées par approximations successives. À chaque étape, la méthode du gradient descendant[NG 2015] calcule le coût de la nouvelle fonction hypothèse et tente de minimiser celle-ci. Cette minimisation est possible à la condition que la fonction de coût soit différentiable. C'est, en effet, en dérivant la fonction de coût qu'un ajustement est calculé en vue d'une réduction.

[NG 2015] Ng Andrew, *CS229 Lecture notes : Supervised learning*, http://cs229.stanford.edu/notes/cs229-notes1.pdf, 2015, consulté le 15 octobre 2016

Mathématiquement, le vecteur θ_j optimal (l'ensemble des multiplicateurs de la fonction d'hypothèse) se détermine comme suit :

$$Répèter\ pour\ chaque\ variable\ \theta\ \{$$

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1, \dots, \theta_n)$$

$$\}\ (simultanément\ pour\ chaque\ j = 0, 1, \dots, n)$$

où le signe := correspond à une affectation et non à une égalité mathématique.

Pratiquement, l'algorithme du gradient s'explique comme suit :

1. L'algorithme calcule la dérivée de la fonction de coût de la fonction d'hypothèse.
2. Si la norme[8] de la fonction de la dérivée du coût est égale ou inférieure à une valeur de seuil déterminée (ou si un nombre d'itérations déterminé au préalable est atteint dans certaines implémentations), l'algorithme s'interrompt. C'est que la fonction se trouve à proximité immédiate d'un minimum et que tout déplacement complémentaire (modification des variables du modèle) présente peu d'intérêt.
3. Si la dérivée est, à l'inverse, supérieure à cette valeur de seuil, le pas de déplacement dans le sens $-J'(\theta_0, \theta_1, \dots, \theta_m)$ où $J'(\theta_0, \theta_1, \dots, \theta_m)$ désigne la fonction dérivée de la fonction $J(\theta_0, \theta_1, \dots, \theta_m)$. Le pas de la fonction de coût est proportionnel à la valeur de cette dérivée et sera donc plus ou moins important en direction du minimum.
4. On calcule l'itération suivante comme suit : $J_{k+1} = J_k - \alpha J'(\theta_0, \theta_1, \dots, \theta_m)$ où k représente le numéro de l'itération actuelle.

Une traduction de cet algorithme en Python se trouve en annexe (voir IV.G Traduction du gradient descendant en Python).

Le calcul de la dérivée de la fonction de coût indique la tangente de la fonction avec le vecteur θ dans l'état actuel et donc la direction du pas à effectuer pour s'approcher du minimum. Le

[8] Valeur absolue appliquée à un vecteur

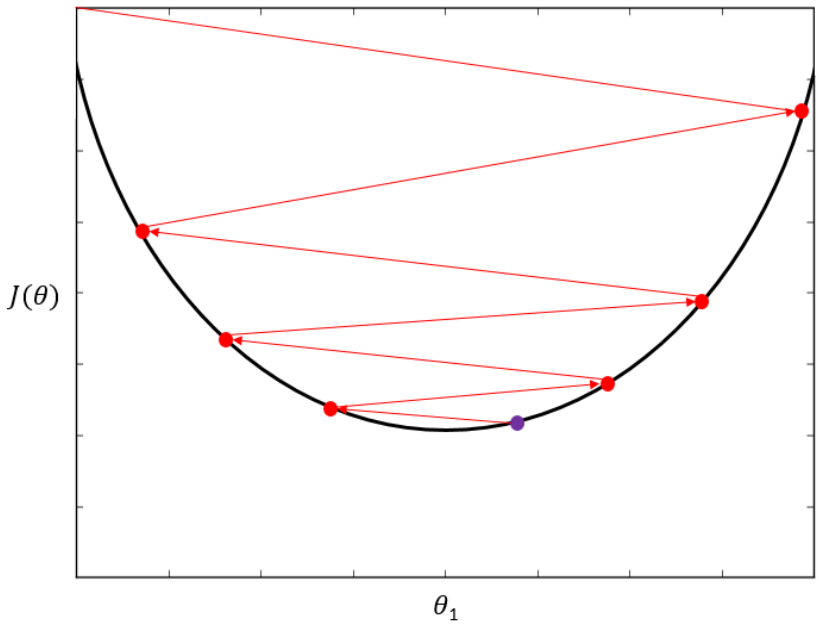
pas s'ajuste proportionnellement à la dérivée pour devenir de plus en plus précis c'est-à-dire en réduisant le déplacement quand la norme de la dérivée diminue.

Ce pas doit être ajusté manuellement en observant les premières exécutions de l'algorithme. C'est pourquoi il est multiplié par le taux d'apprentissage (*learning rate*), noté α ci-dessus.

Le taux d'apprentissage (*learning rate*) est la seule variable qui doit être définie initialement. Sa fixation influencera fortement le résultat ou la performance de la régression.

Si sa valeur est trop élevée, l'algorithme du gradient descendant pourrait s'éloigner du minimum car le pas calculé à chaque itération serait trop important. Lors de la minimisation, le coût augmenterait alors jusqu'à l'interruption de l'algorithme.

L'image suivante illustre ce phénomène. Si la valeur initiale de θ est symbolisée par le point mauve, un taux d'apprentissage fixé à une valeur trop importante engendre le franchissement du minimum en s'en éloignant. Même si chaque pas suivant se fait dans la direction adaptée, le coût calculé ne cesserait de croître.

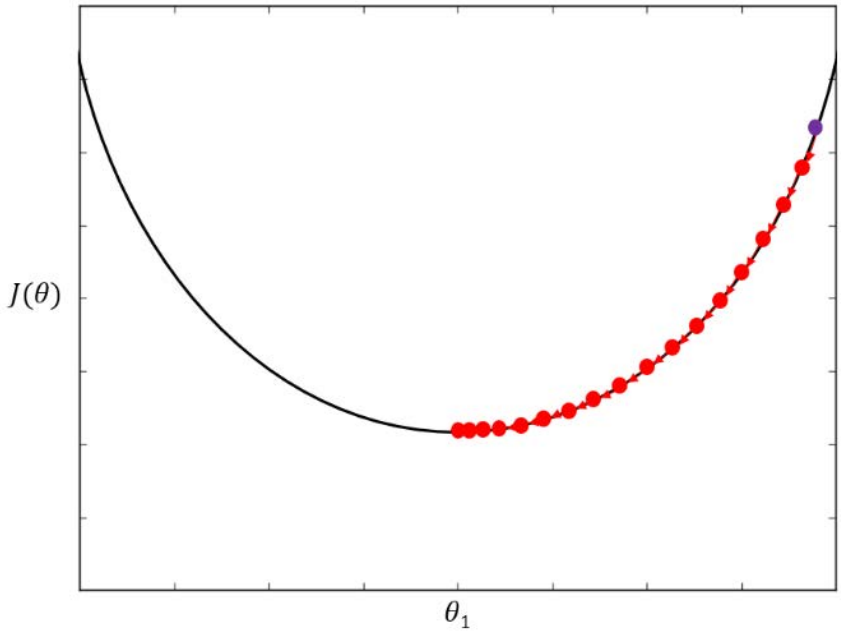


Algorithme du gradient avec un taux d'apprentissage excessif, 1.

Si à l'inverse, le taux d'apprentissage est trop faible, l'algorithme réalisera un très grand nombre de calculs intermédiaires avant de converger. Cela implique que celui-ci consommera

davantage de ressources sur le matériel que ce qui est raisonnable. Théoriquement, cela ne nuit donc qu'à la vitesse d'exécution.

En pratique, le calcul du gradient descendant pourrait s'avérer irréalisable sur la machine hôte choisie, la quantité de mémoire pouvant s'avérer insuffisante.



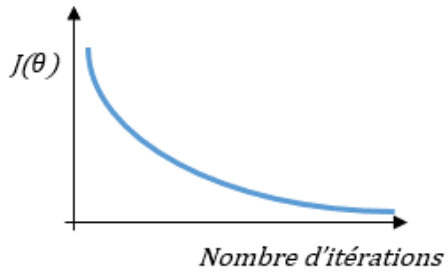
Algorithme du gradient avec un taux d'apprentissage insuffisant, 1. [image inspirée d'une esquisse du cours en ligne de Coursera.org (Introduction to Machine Learning), 2015]

Il est donc important de tester et évaluer différents taux d'apprentissage. S'il n'existe pas de formule mathématique particulière permettant de calculer celui-ci, les travaux[NG 2015a] de l'Université de Stanford menés par Andrew Ng indiquent qu'il soit judicieux de fixer le taux d'apprentissage à une valeur faible et de l'ajuster ensuite. Celui-ci pourrait, par exemple, être fixé à 0.01 et être multiplié ou divisé approximativement par 3 au regard du comportement observé (0.0003 <- 0.001 <- 0.003 <- 0.01 -> 0.03 -> 0.1 -> 0.3 -> 1).

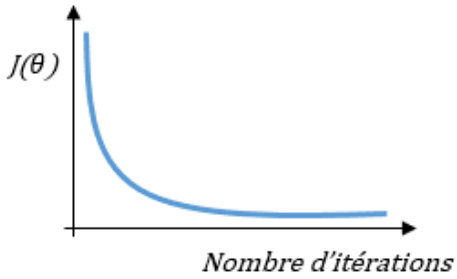
En surveillant l'évolution du coût à chaque étape, il est possible de tirer la conclusion adéquate afin de déterminer une valeur acceptable.

Idéalement, le coût doit décroître rapidement dès les premières itérations.

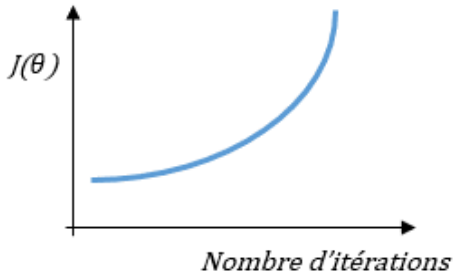
[NG 2015a] Ng Andrew, *CS229 Lecture notes : Supervised learning*, http://cs229.stanford.edu/notes/cs229-notes1.pdf, 2015, consulté le 15 octobre 2016



S'il décroît lentement, le taux d'apprentissage devrait être rehaussé pour accélérer l'algorithme.



Si le coût chute rapidement, le taux d'apprentissage semble adapté.



A l'inverse, si le coût augmente le taux d'apprentissage doit être diminué.

Evolution du coût – différents cas de figure, 1.

L'exécution du gradient descendant très coûteuse en ressources et temps de calcul car il additionne le coût de chaque données observées à chaque pas ! En pratique, des alternatives « allégées » (moins coûteuses en ressources, tout en restant efficaces) comme le gradient descendant stochastique ou mini-batch le remplacent dans des applications concrètes.

2. Le gradient descendant stochastique (Stochastic gradient descent)

Le gradient descendant stochastique[NG 2015a] s'avère beaucoup plus rapide que l'algorithme classique du gradient descendant tout en restant très proche de ce dernier conceptuellement. Il est employé exclusivement lorsque la durée de l'apprentissage ou la quantité de mémoire à disposition sont des contraintes fortes, généralement parce que le jeu de données est très

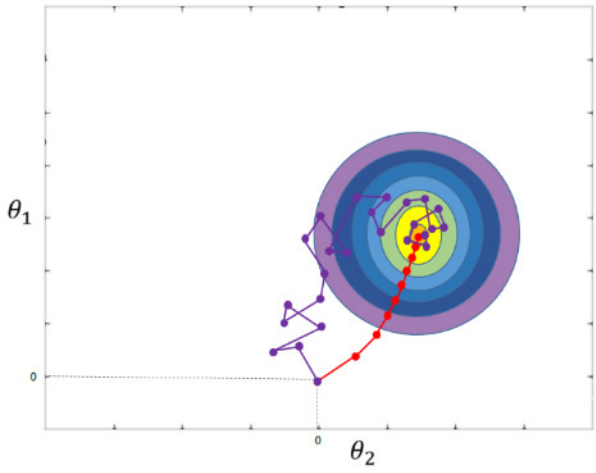
[NG 2015a] Ng Andrew, *CS229 Lecture notes : Supervised learning*, http://cs229.stanford.edu/notes/cs229-notes1.pdf (LMS Algorithm), 2015, consulté le 15 octobre 2016

volumineux. Les expérimentations[BOTTOU 2012] menées par L. Bottou pour Microsoft Research suggèrent, par exemple, son utilisation pour l'entrainement des réseaux neuronaux.

Le gradient descendant stochastique se veut moins rigoureux dans l'évaluation du pas à franchir en limitant son calcul du déplacement à chaque itération à une seule donnée. Le jeu d'entrainement est donc parcouru une seule fois dans son intégralité et non plus à chaque pas comme dans sa version originale.

Par conséquent, le déplacement engendré lors d'une itération peut s'éloigner du minimum. En pratique, on remarque que cet algorithme donne bien souvent de très bon résultat et qu'il termine son exécution en oscillant à proximité immédiate du minimum.

En préparation à son exécution, il est recommandé de mélanger les données d'exemples aléatoirement. Cela réduit le risque de voir le gradient faire plusieurs pas successifs dans la même direction.



Représentation schématique de l'évolution du coût avec l'algorithme du gradient descendant (en rouge) et avec le gradient stochastique (en mauve), 1. [image inspirée d'une esquisse du cours en ligne de Coursera.org (Introduction to Machine Learning), 2015]

3. Le gradient descendant mini-batch (*Mini-Batch Gradient Descent*)

Le gradient descendant mini-batch est une version intermédiaire du gradient descendant original et de sa version stochastique. Il réalise la minimisation de l'hypothèse par lot de

[BOTTOU 2012] Bottou, L., *Stochastic Gradient Descent Tricks*, http://research.microsoft.com/pubs/192769/tricks-2012.pdf, 1-9 p., 2012, consulté le 10 avril 2016

données à usage unique. Les données d'entrainement sont alors segmentées en plusieurs lots de de données de même taille et utilisés successivement comme dans l'algorithme de gradient descendant stochastique.

Cette implémentation est généralement plus efficace que sa variante stochastique car celle-ci, pour peu qu'elle soit vectorisée, autorise le parallélisme des opérations de calcul.

Cet algorithme, comme les autres versions du gradient descendant, souffre cependant également des mêmes maux[QUOC 2011]. Il nécessite l'ajustement du taux d'apprentissage manuellement pour être efficace. Cela complexifie donc légèrement le travail de développement et de mise à l'échelle de la solution.

4. Les algorithmes BFGS, L-BFGS et gradient conjugué

Les algorithmes BFGS, L-BFGS et gradient conjugué[QUOC 2011] sont trois méthodes possédant l'avantage de ne pas devoir être impérativement paramétrées par un taux d'apprentissage. Ces méthodes ont la capacité d'être parallélisées comme les méthodes de gradient descendant stochastique et mini-batch.

Brièvement, la technique BFGS a été mise au point[9] par l'utilisation de la méthode de Quasi-Newton dans l'algorithme d'optimalisation. Cette dernière remplace l'opération de dérivée (permettant de suivre la pente descendante et d'en calculer le pas dans les méthodes de gradient descendant) par une approximation plus simple à calculer.

De manière analogue, la méthode du gradient conjugué réalise ces approximations par un mécanisme différent, non-étudié dans ce document, mais similaire sur son utilité.

Par modifications successives des paramètres de la fonction d'hypothèse grâce à ces approximations, la minimisation est nettement moins coûteuse en ressources.

[QUOC 2011] Quoc V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, A. Ng, *On optimization methods for deep learning*, http://cs.stanford.edu/~acoates/papers/LeNgiCoaLahProNg11.pdf, 2011, consulté le 10 avril 2016

[9] La méthode BFGS a été nommée avec les initiales de ses inventeurs Broyden, Fletcher, Goldfarb et Shanno en 1970.

La variante L-BFGS est une version optimisée de la méthode BFGS. Celle-ci veille à limiter la consommation de mémoire nécessaire à l'entrainement par segmentation du lot de données en lots plus petits comme le fait la variante mini-batch du gradient descendant.

La méthode BFGS a, en outre, la capacité à être optimisée avec de nouvelles données sans devoir relire l'ensemble du jeu d'entrainement car celui-ci conserve l'information utile pour normaliser de nouvelles données.

Les expérimentations présentées dans la publication « *On optimization methods for deep learning* » citée plus haut, recommandent l'utilisation de L-BFGS qui donne généralement de meilleurs résultats pour les problèmes de petites dimensions et la méthode du gradient conjugué pour les problèmes à grandes dimensions (dont les données possèdent de très nombreuses caractéristiques).

D. Evaluation de l'efficacité de l'algorithme d'apprentissage

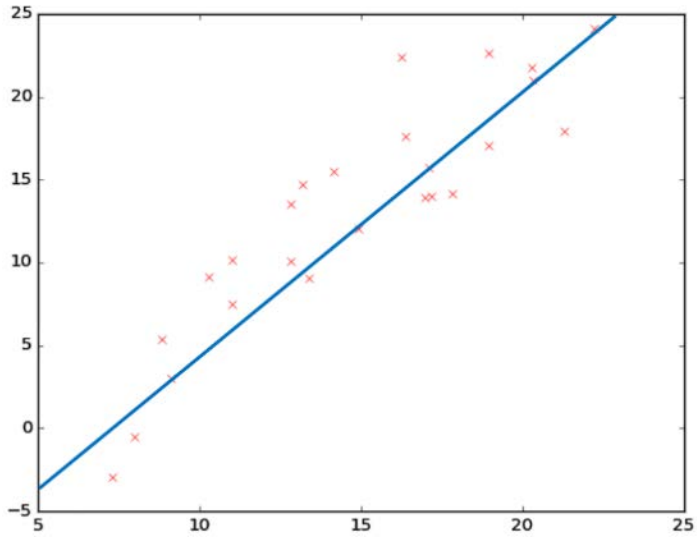
L'efficacité d'un algorithme de régression ne doit pas être évaluée sur sa seule capacité à déterminer une fonction réduisant le coût au minimum. Sa capacité à généraliser (c'est-à-dire à rester en phase avec le concept qu'elle représente) est aussi primordiale. On considère généralement que la fonction la plus simple à un coût raisonnable est la meilleure option.

1. Le biais (*Bias*) et la variance

Une fonction d'hypothèse peut être caractérisée comme étant plus ou moins biaisée (*high bias*) ou plus ou moins variante (*high variance*)[JAMES 2015b].

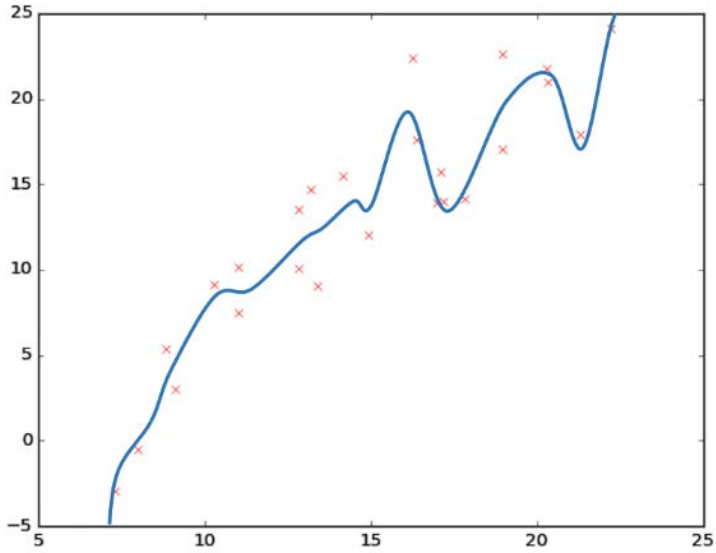
Le biais est la tendance à généraliser, c'est-à-dire à simplifier, parfois excessivement.

[JAMES 2015b] James G., Witten D., Hastie T., Tibshirani R., *An Introduction to Statistical Learning*, 2e édition, New York : Springer, 2015, 33-36 p. (2.2.2 The Bias-Variance Trade-Off)



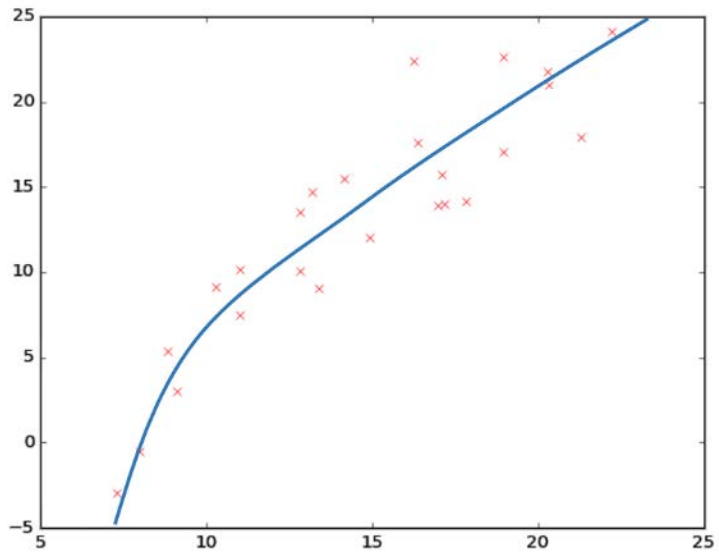
Régression linéaire, 2. [données publiées par le programme de développement des Nations Unies (http://hdr.undp.org/fr/data), 2015]

A l'inverse, la variance représente la propension à suivre excessivement les données et donc, parfois à généraliser insuffisamment. Ce phénomène engendre bien souvent une représentation trop complexe et volatile du domaine à représenter.



Régression non-linéaire, 3. [données publiées par le programme de développement des Nations Unies (http://hdr.undp.org/fr/data)]

Une bonne hypothèse n'est donc pas la fonction dont le coût avec le jeu de données est le plus faible mais une fonction dont le coût est limité tout en limitant au maximum sa complexité.



Régression non linéaire, 4. [données publiées par le programme de développement des Nations Unies (http://hdr.undp.org/fr/data), 2015]

2. Phases d'évaluation de l'algorithme d'apprentissage automatique

Pour entraîner une fonction d'hypothèse, il est recommandé[NG 2015b] de ne pas utiliser l'ensemble des données à sa disposition. De manière générale, on segmente les données en deux parties : 70 % pour le jeu d'entrainement (*training set*) et 30% pour le jeu de test (*test set*). Ces ensembles vont nous permettre de confronter nos fonctions hypothèses et d'en évaluer leur précision. Cela est particulièrement utile lorsque le nombre de dimensions des données ne permettent pas de les visualiser graphiquement.

Si le coût $J(\theta)$ à l'entrainement (*training set error*) est faible alors qu'il est important lors du test (*test set error*), c'est que la variance est trop importante (on parle alors d'*overfitting*). L'idéal est donc que le coût calculé avec le jeu de test approche celui du jeu d'entrainement.

Lorsqu'on segmente les données en deux ensembles, il n'est cependant plus possible de déterminer le taux d'erreur résiduel car toutes les données ont été utilisées ; c'est pourquoi, on suggère parfois de segmenter en trois ensembles plutôt qu'en deux. On répartit alors les

[NG 2015b] Ng Andrew, *CS229 Lecture notes : Regularization and model selection (Cross validation)*, http://cs229.stanford.edu/notes/cs229-notes5.pdf, 2015, consulté le 21 octobre 2016

données comme suit : 60% pour le *training set*, 20% pour le *cross validation set*, 20% pour le *test set*[10].

3. Indicateurs de performance

Lors d'une classification, trois indicateurs étaient communément utilisés pour rendre compte de la performance d'une hypothèse[GOUTTE 2005].

- La précision ;
- Le rappel (*recall*) ;
- Le score F_1.

Ces indicateurs distinguent quatre cas de figure lors d'une classification d'une donnée.

Celle-ci est :

- Un cas positif évalué positivement (vrai positif);
- Un cas négatif évalué négativement (vrai négatif);
- Un cas négatif évalué positivement (faux positif) ;
- Un cas positif évalué négativement et donc incorrectement (faux négatif).

		Appartenance réelle	
		Vrai	Faux
Prédiction	Vrai	Vrai positif	Faux positif
	Faux	Faux négatif	Vrai négatif

Classification des prédictions avec une matrice de confusion

[10] Ce principe est illustré par un exemple dans la section relative aux régressions non linéaires dans la section II.D.2.

[GOUTTE 2005] Goutte C., Gaussier E., *A Probabilistic Interpretation of Precision, Recall and F-score, with Implication for Evaluation*, http://www3.xrce.xerox.com/content/download/20797/148382/file/xrce_eval.pdf, 2005, consulté la 10 avril 2016 – Ce travail reprend le détail des indicateurs proposés initialement par Van Rijsbergen en 1975

a) La précision

La précision se caractérise par l'étude des cas évalués positivement. Elle établit la proportion de données réellement positives au prorata de toutes celles déclarées comme telles.

$$Precision = \frac{Vrais\ positifs}{Nombre\ de\ cas\ évalués\ positivement} = \frac{Vrais\ positifs}{Vrais\ positifs + Faux\ positifs}$$

L'indicateur de précision mesure donc la capacité à n'évaluer positivement que les cas réellement positifs uniquement.

b) Le rappel

Le rappel détermine le nombre d'erreurs de classification sur les cas réellement positifs.

$$Rappel = \frac{Vrais\ positifs}{Nombre\ de\ cas\ évalués\ correctement} = \frac{Vrais\ positifs}{Vrais\ positifs + Faux\ négatifs}$$

L'indicateur de rappel indique donc la capacité à retrouver tous les cas réellement positifs.

De manière trompeuse, cet indice atteindrait donc 100% pour tout algorithme renseignant toutes les données évaluées comme positives.

Dans certaines applications, la précision est cruciale et le rappel peu important mais il existe également des domaines dans lesquels c'est l'inverse. La précision est, par exemple, cruciale dans la sélection d'un algorithme de recommandation où seules quelques suggestions parmi les résultats sont utiles. Celles-ci se doivent donc d'être correctes !

A l'inverse, si, un algorithme doit déterminer quels patients sont à risque pour une pathologie particulière, le rappel doit être privilégié afin de n'ignorer aucun cas de patient potentiellement positif.

c) Le score F1

Le score F_1 à pour objectif de retourner un indicateur unique pour simplifier la comparaison de plusieurs modèles. Celui-ci utilise les indices de précision et de rappel pour retourner une information unique de la performance.

$$F_1 Score = 2\frac{P * R}{P + R}$$

où P est l'indice de précision et R l'indice de rappel.

Cet indice est régulièrement utilisé pour choisir l'algorithme qui sera retenu dans un objectif précis. Il permet, en effet, d'étudier le taux d'erreur indépendamment du type d'algorithme testé. Le plus haut score obtenu indique l'algorithme le plus performant.

Tableau 1 : Exemples de scores F_1 calculés :

	Précision	Rappel	Score F1
Algorithme 1	80%	20%	32%
Algorithme 2	90%	40%	55%
Algorithme 3	50%	70%	58%
Algorithme 4	70%	40%	51%
Algorithme 5	35%	80%	49%

Exemples de score F_1 réalisés au départ de données fictives.

Ce score est donc déterminé dans un intervalle de 0 à 100%, un résultat de 100% signalant que la précision et le rappel sont maximaux. Cet indicateur souffre, par contre, d'un problème d'expressivité car celui-ci ne rend pas compte de la classification correcte d'une donnée négativement (vrai négatif).

d) L'indice ROC

L'alternative ROC[POWDERS 2011], *Receiver Operating Characteristics*, offre une autre manière de juger de la performance de classification au moyen d'un indicateur comprenant deux valeurs : la sensibilité et la spécificité.

(1) La sensibilité

La sensibilité est la probabilité que l'algorithme de classification retourne une valeur positive sur les cas effectivement positif. Celle-ci se calcule comme suit :

$$Sensibilité = \frac{Vrais\ positifs}{Vrais\ positifs + Faux\ négatifs}$$

[POWDERS 2011] Powders D. M. W., *Evaluation : From Precision, Recall and F-Measure to ROC Informedness, Markness & Correlation*, http://www.bioinfopublication.org/files/articles/2_1_1_JMLT.pdf, 2011, consulté le 9 avril 2016

(2) La spécificité

La spécificité est la probabilité que l'algorithme perçoive négativement un cas réellement négatif. Celle-ci se calcule comme suit :

$$Spécificité = \frac{Vrais\ négatifs}{Vrais\ négatifs + Faux\ positifs}$$

Ces deux indicateurs sont fixés dans l'intervalle [0,1].

Prise séparément ces indicateurs non pas de sens, c'est pourquoi ils sont notés au sein de la même parenthèse.

Le score (1,1) désigne un classificateur n'ayant aucun faux négatif ou positif et donc un classificateur parfait. A l'inverse, le score (0,0) désigne un classificateur dont le taux d'erreur est de 100%.

Powders perçoit cette méthode comme élégante car celle-ci tient compte de l'ensemble des résultats.

e) Le nombre d'erreurs de classification

Lorsqu'il est question d'évaluer des algorithmes de classification où le nombre de classes est supérieur à deux, on parle de perte (*loss*) ou du nombre de classifications erronées (*misclassification errors*). Nommé en anglais « accuracy », cet indicateur n'est pas un indice de précision. Il accumule le nombre d'erreurs de classification globalement pour toutes les classes pour l'ensemble des données testées sans faire la distinction entre faux positifs et faux négatifs.

III. Algorithmes

A. Types d'apprentissage

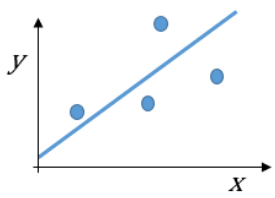
Il existe deux familles d'algorithmes d'apprentissage automatique suivant que l'apprentissage soit supervisé ou non.

L'apprentissage supervisé a pour objectif de définir une représentation mathématique d'une application connue pour réaliser une prédiction ou une classification. Il se détermine au départ d'exemples dont le résultat est connu.

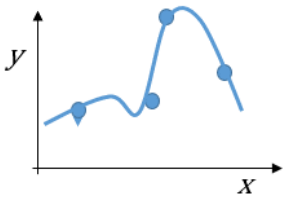
A l'inverse, l'apprentissage non supervisé tente de mettre en évidence des partitionnements dans des données dont on ignore encore la classification. Cet apprentissage essaiera, dès lors, de regrouper les exemples par l'analyse de leurs attributs disponibles. Il appartiendra ensuite à l'analyste d'en déduire le sens.

B. Algorithmes d'apprentissage supervisé

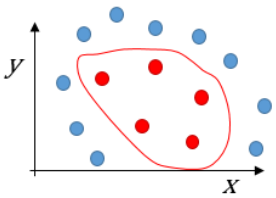
Il existe de nombreuses méthodes d'apprentissage supervisé tels que les régressions et les classifications.



Régression linéaire



Régression non linéaire



Régression logistique ou classification

Différentes méthodes appartenant aux apprentissages supervisés.

Les algorithmes de régression linéaire et non linéaire définissent une fonction mathématique représentative de la relation d'une variable par rapport à une ou plusieurs autres. Ils permettent ainsi de prédire ensuite cette variable au regard de données nouvelles.

A l'inverse, les algorithmes de classification tentent de déterminer la frontière entre des données classifiées différemment. En d'autres mots, là où les algorithmes de régression tentent de

prédire une variable quantitative, les algorithmes de classification tente de déterminer une variable qualitative.

1. Régression linéaire (*Linear regression*)

Le type de modélisation le plus simple est la régression linéaire[JAMES 2015c] ce qui signifie que son résultat est une fonction modélisant une droite qui minimise l'écart avec les données d'entrée.

$$h_\theta(x) = \theta_0 + \theta_1 x$$

Son calcul peut être réalisé par la méthode des moindres carrés en une opération unique. Le vecteur *Theta* (noté θ), comprenant les multiplicateurs $\theta_0$et $\theta_1$tels que la fonction $f(x) = \theta_0 + \theta_1 x$ est minimalisée, se calcule directement comme suit :

$$\theta = (X^T X)^{-1} X^T y$$

où X est la matrice contenant les données d'entrée de dimension m (le nombre de données) x n (le nombre de variables par donnée) et y les résultats connus associés à ces données. La notation $()^{-1}$ signifie que la matrice A est inversée tel que $A\,A^{-1} = I_n$, où $I_n$est la matrice d'identité de degré n.

Si $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ et $ad - bc \neq 0$ alors A est inversible et $A^{-1} = \frac{1}{ad-bc}\begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$. La méthode de Gauss permet de réaliser ces inversions pour les matrices d'autres dimensions.

Le calcul de $(X^T X)^{-1}$ est cependant très coûteux si n est important! Il donne lieu à l'inversion d'une matrice $n \times n$ dont on évalue la complexité de l'ordre de $O(n^3)$. Dans certain cas, par exemple lorsqu'une colonne est multiple d'une autre, l'inversion peut même s'avérer impossible ! C'est également le cas lorsque nombre de variables par données est trop important.

a) Illustration par un exemple

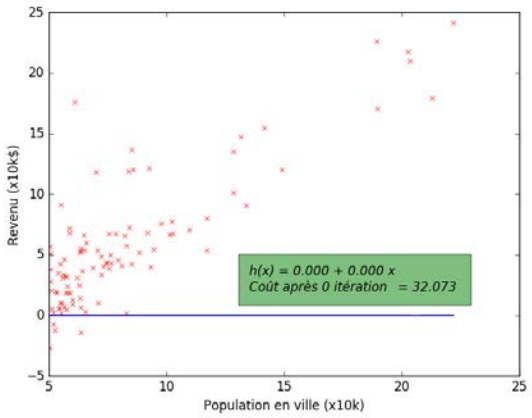
L'exemple suivant illustre l'évolution par l'algorithme du gradient au départ d'un jeu de données[11] d'une seule variable, c'est-à-dire que pour une valeur donnée, notée x, correspond une valeur de prédiction unique, notée y telle que $h(x) = y$.

[JAMES 2015c] James G., Witten D., Hastie T., Tibshirani R., *An Introduction to Statistical Learning*, 2e édition, New York : Springer, 2015, 61-90 p. (3.1 Simple Linear Regression)

[11] Jeu de données mis à disposition librement sur Internet : https://github.com/madisonmay/python-ml-class/tree/master/ex1

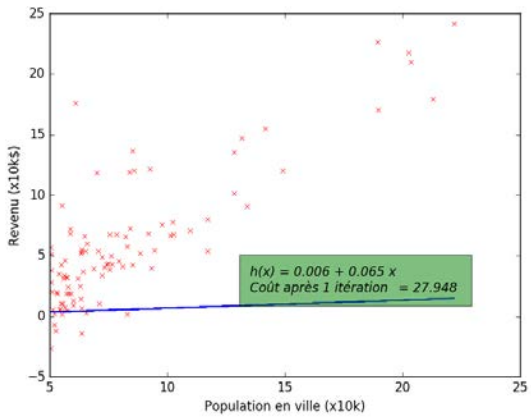
Ce jeu de données est mis librement à disposition par l'université de Stanford et illustre l'évolution du revenu annuel moyen par habitant en fonction de la population de la ville où il s'est établi. Pour l'exercice, seules les villes de plus de 50000 habitants ont été retenues. Les données sont enregistrées au format où le premier nombre représente la population en ville et le second le revenu moyen.

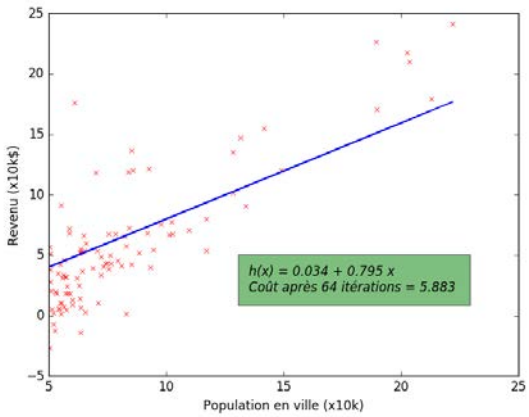
Après chargement des données, l'algorithme s'initialise avec les valeurs $\theta_0 = \theta_1 = 0$.
L'hypothèse
$h_\theta(x) = \theta_0 + \theta_1 x = 0$ se représente comme suit :



Régression linéaire, 2. [données remises par l'université de Stanford à des fins pédagogiques sur Coursera.org (Introduction to Machine Learning), 2015]

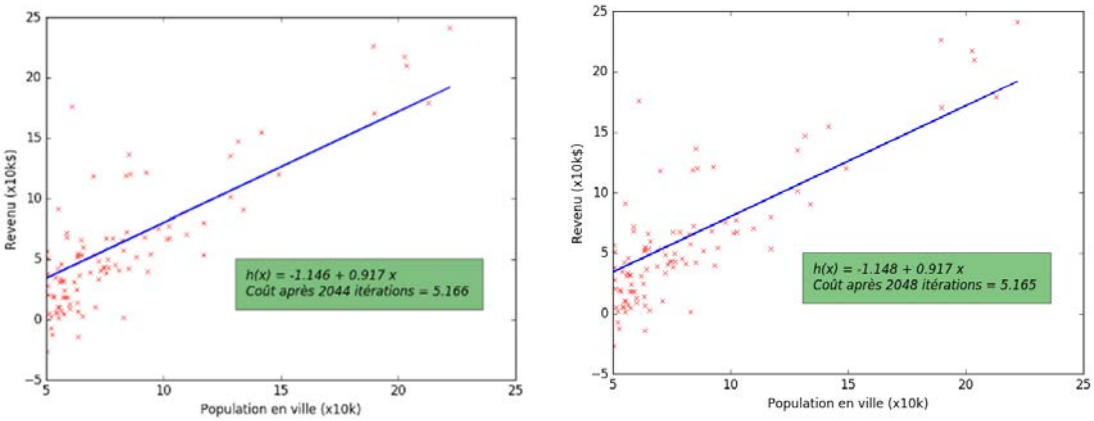
Le coût est alors calculé une première fois et dérivé. Dès la première itération, les multiplicateurs de la fonction d'hypothèse évoluent pour réduire le coût qui correspond à la somme des écarts entre les données et l'hypothèse.





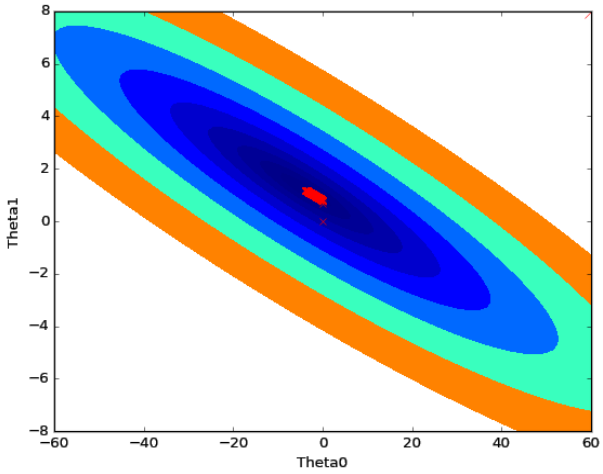
Régression linéaire, 3, 4. [données remises par l'université de Stanford à des fins pédagogiques sur Coursera.org (Introduction to Machine Learning), 2015]

Lors des dernières itérations, le coût évolue très peu. C'est un signe que l'algorithme est sur le point d'atteindre un minimum de la fonction de coût.



Régression linéaire, 5, 6. [données remises par l'université de Stanford à des fins pédagogiques sur Coursera.org (Introduction to Machine Learning), 2015]

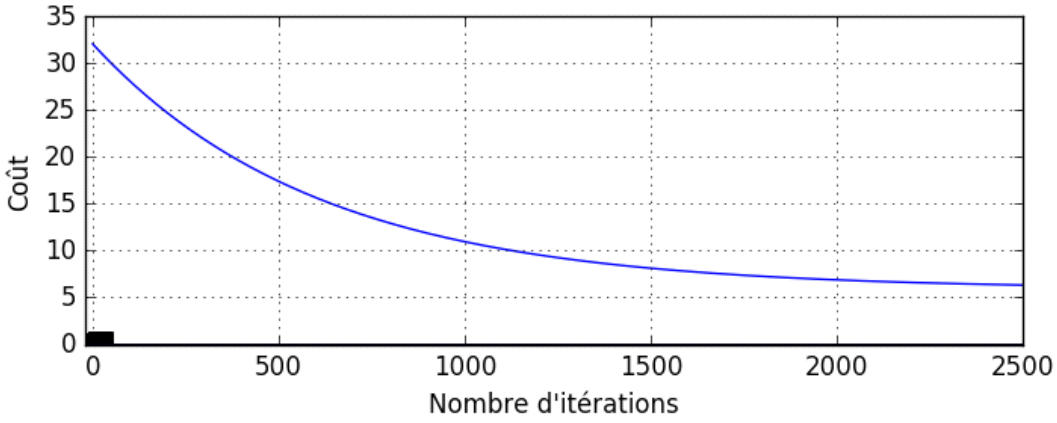
Vu sous un autre angle, celui de l'évolution du coût, on remarque que celui-ci s'approche du minimum à chaque itération.



Evolution du coût au regard de l'ajustement du vecteur theta, 1. [données remises par l'université de Stanford à des fins pédagogiques sur Coursera.org (Introduction to Machine Learning), 2015]

Les nuances de couleurs expriment le coût, du plus élevé en blanc au plus faible en bleu foncé. Les marques rouges indiquent quels sauts intermédiaires ont été réalisé par l'algorithme de minimisation.

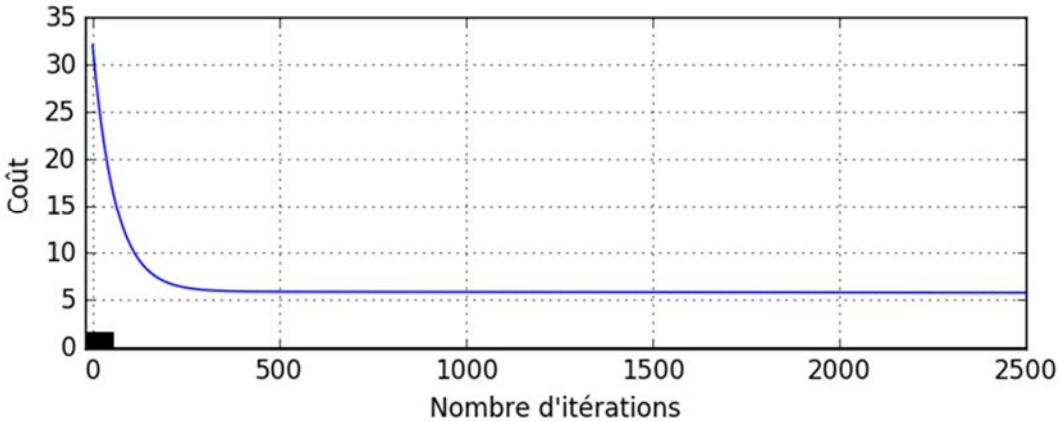
Par l'analyse de la diminution du coût à chaque itération, on peut conclure que le gradient descendant fonctionne correctement car celui est décroissant. Celui-ci diminue cependant lentement. Il est donc judicieux de procéder à l'ajustement du taux d'apprentissage.



Evolution du coût en fonction du nombre d'itérations, 1. (2016). [données remises par l'université de Stanford à des fins pédagogiques sur Coursera.org (Introduction to Machine Learning), 2015]

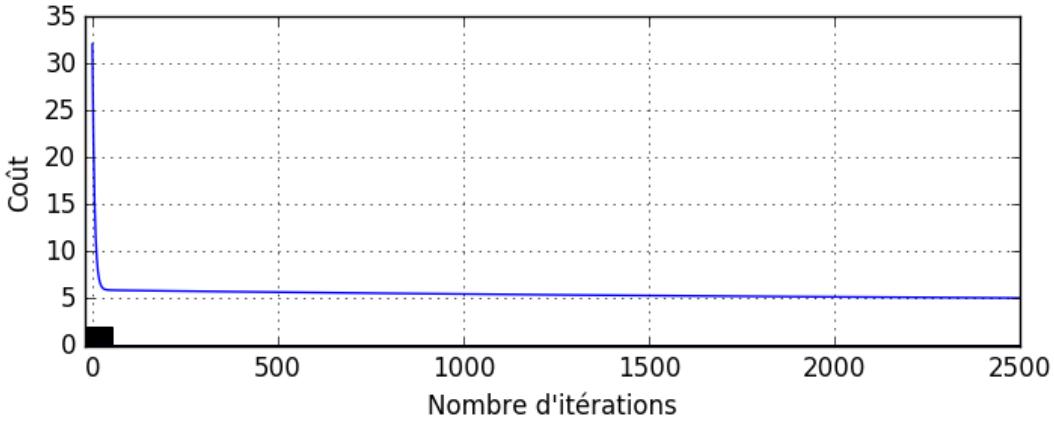
En l'augmentant à plusieurs reprises par multiplication par trois, nous obtenons rapidement un algoritme qui approche beaucoup plus rapidement le minimum. Cet ajustement a pour objectif d'éviter des calculs superflus et de réduire, par conséquent, le temps et les ressources nécessaires à son exécution.

Ici, multiplié à deux reprises par trois, le taux d'apprentissage force une diminution du coût beaucoup plus nette dès les premières itérations.



Evolution du coût en fonction du nombre d'itérations, 2. [données remises par l'université de Stanford à des fins pédagogiques sur Coursera.org (Introduction to Machine Learning), 2015]

Multiplié une nouvelle fois à deux reprises par trois, le taux d'apprentissage est optimal (Au-delà, l'algorithme disfonctionne en s'éloignant du minimum.).



Evolution du coût en fonction du nombre d'itérations, 3. [données remises par l'université de Stanford à des fins pédagogiques sur Coursera.org (Introduction to Machine Learning), 2015]

Le résultat de cet apprentissage retourne la fonction $h(x) = -1{,}148 + 0{,}917\,x$. Celle-ci peut alors nous permettre de prédire le revenu moyen annuel par habitant au regard de la population de la ville où celui-ci vit.

Cet exemple est particulièrement simpliste et ignore les autres caractéristiques qui permettraient de réaliser un modèle plus rigoureux. On pourrait, par exemple, s'interroger sur le fait que le critère de population soit pertinent et que d'autres métriques comme l'activité économique, le taux d'emploi, le niveau d'étude ne soient pas prises en compte.

On pourrait également d'interroger sur le postulat fait que notre fonction d'hypothèse épouse la forme d'une droite, comme s'il existait une relation proportionnelle entre ces variables et son résultat. Il s'agit d'une hypothèse bien trop forte[JAMES 2015d] pour la plupart des représentations.

En pratique, les régressions linéaires sont souvent délaissées au profit des régressions non linéaires qui s'ajustent plus correctement aux différents domaines d'application.

[JAMES 2015d] James G., Witten D., Hastie T., Tibshirani R., *An Introduction to Statistical Learning*, 2e édition, New York : Springer, 2015, 104 p.

2. Régression non linéaire (*Non linear regression*)

La régression non linéaire[JAMES 2015e] permet de modéliser des fonctions plus complexes que la régression linéaire. Elle tente d'épouser plus fidèlement les données d'entrée pour limiter le coût et obtenir ainsi une fonction plus cohérente.

L'idée est donc d'entrainer notre algorithme au moyen d'une fonction hypothèse plus complexe et de faire varier ses paramètres comme lors d'une régression linéaire par le biais d'un algorithme comme celui du gradient descendant. C'est en effet en ajoutant certains paramètres exposés à différentes puissances que la fonction résultante courbera et approchera les données.

a) Régression non linéaire avec un seul attribut

L'exemple suivant illustre une régression non linéaire au départ de données de publication annuelle de l'Organisation des Nations Unies comparant l'indice de développement humain dans tous les pays membres pour l'année 2015[12]. Seules deux caractéristiques sont ici utilisées : l'espérance de vie et le revenu annuel par habitant. Le postulat est qu'il est possible de mettre en évidence une relation entre l'espérance de vie et le revenu annuel moyen et donc d'entraîner l'hypothèse pour permettre d'estimer le revenu annuel moyen.

Sa représentation mathématique prend la forme d'une fonction polynomiale où l'espérance de vie est exposée à différentes puissances.

Polynôme de degré 1 : $h_\theta(x) = \theta_0 + \theta_1 x$

Polynôme de degré 2 : $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$

…

Polynôme de degré n : $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_n x^{n-1}$

L'objectif poursuivi est de mettre en évidence la difficulté de déterminer quel degré est idéal pour généraliser au mieux.

[JAMES 2015e] James G., Witten D., Hastie T., Tibshirani R., *An Introduction to Statistical Learning*, 2e édition, New York : Springer, 2015, 90-95, 266, 277-280 p.

[12] Données accessible à l'adresse suivante : http://hdr.undp.org/fr/data

Comme dans l'exemple précédent, les données se présentent sous forme de fichier plat au format csv.

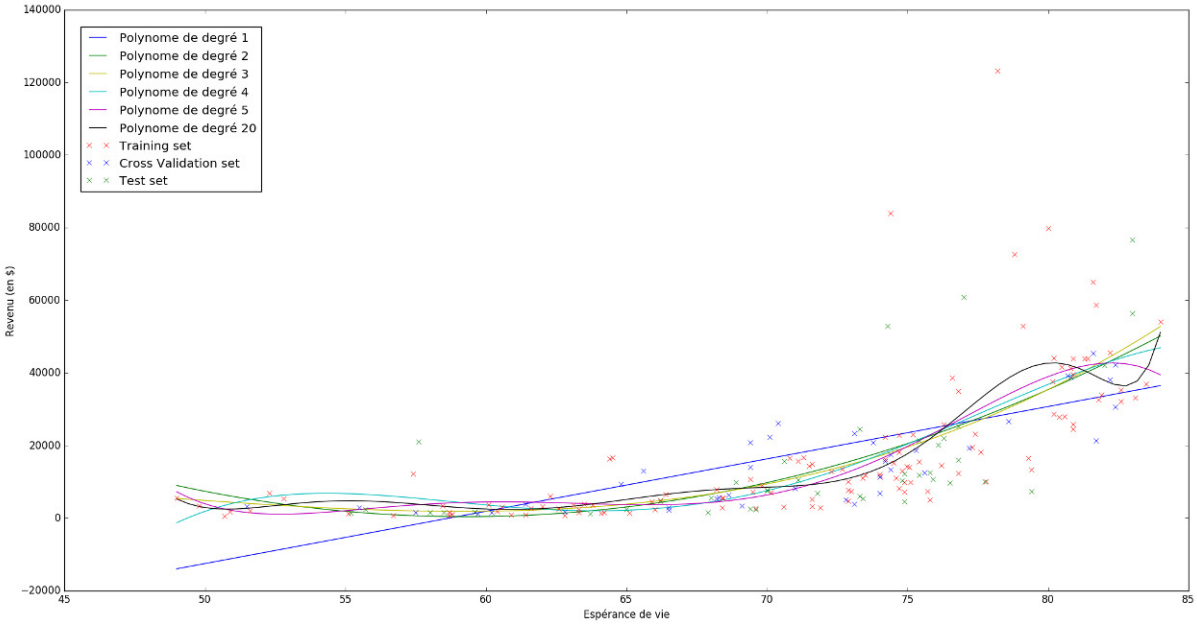
81,6 ; 64992,34046
82,4 ; 42260,61295
83 ; 56431,06833
80,2 ; 44025,48217

Extrait des données d'entrée

Après avoir mélangé les données (initialement, celles-ci sont triées), les données sont réparties en trois ensembles reprenant respectivement 60%, 20% et les 20% restants des données.

L'application de l'algorithme du gradient sur les données d'entrainement retourne une fonction hypothèse optimisée pour chacun des degrés polynomiaux. Celles-ci me permettent de déterminer le vecteur θ minimisant la fonction de coût pour chaque degré polynomial voulu.

Représentée, graphiquement, les fonctions des degrés polynomiaux[13] 1, 2, 3, 4, 5 et 20 obtenues se présentent comme suit :



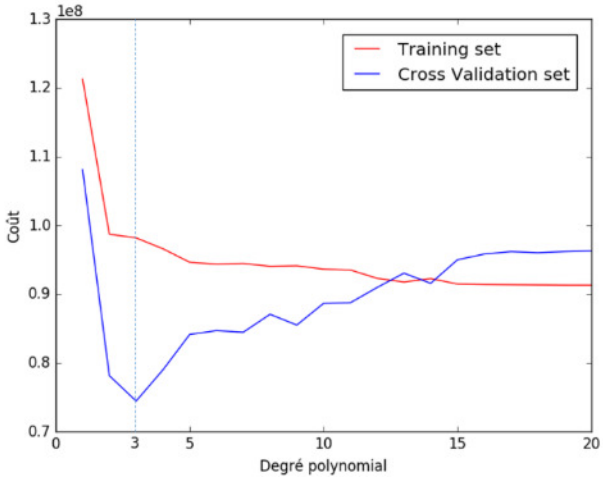
Régression non linéaire - IDH, 1. [données remises par l'Organisation des Nations Unies dans sa publication annuelle « Human Development Statistical Tables » (http://hdr.undp.org/fr/data), 2015]

[13] Script de génération de la régression non linéaire et de la présentation des résultats disponible en annexe

En évaluant le coût global de chacune de ces équations, on remarque que le coût diminue sur le jeu d'entrainement plus le degré polynomial est grand. La comparaison des coûts calculés n'est donc pas un critère satisfaisant pour déterminer quel degré semble le plus adapté.

Tel que discuté plus haut, c'est en calculant le coût avec les données de validation qu'il est possible de déterminer quelle hypothèse permet de généraliser correctement.

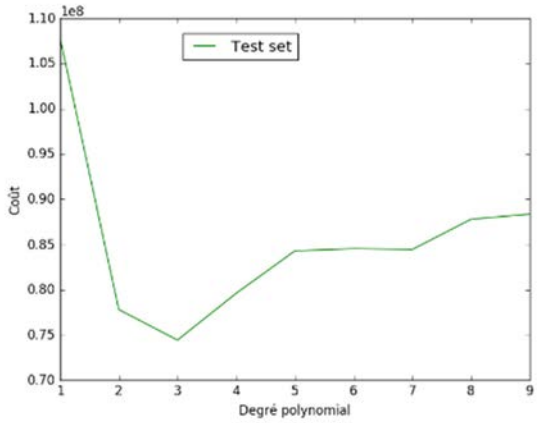
Evalués pour chaque degré de 1 à 20, on observe l'évolution suivante des coûts avec les données d'entrainement et de validation :



Régression non linéaire – Evolution du coût en fonction du degré polynomial - IDH, 2.

Dans cet exemple, le coût est minimal avec un degré polynomial de 3 avec le jeu de données de validation croisée $J_{cv}(\theta)$. C'est donc ce degré qui généralise le mieux les données et s'avère le plus efficace pour prédire de nouvelles données.

Cela est d'ailleurs confirmé par le jeu de données de test qui permet, quant à lui de déterminer le taux d'erreur résiduel.



Régression non linéaire – Evolution du coût en fonction du degré polynomial - IDH, 3.

On retient donc le polynôme de degré 3 avec les valeurs du vecteur θ calculées avec le training set:

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$$

La fonction hypothèse obtenue permet alors de réaliser des prédictions.

$$h_\theta(x) = 1.35572043 - 192.116932\, x + 8505.60915\, x^2 - 10967.0904\, x^3$$

b) Régression non linéaire avec plusieurs paramètres

Il est dans bien des cas utile d'évaluer plusieurs critères pour réaliser une prédiction. La valeur d'une maison, par exemple, est régie par bien plus de critères que sa superficie.

Les fonctions de régression sont capables de prendre en compte plusieurs variables pour déterminer la fonction qui permet d'en prédire la valeur d'un attribut[JAMES 2015f]. De manière analogue à la régression polynomiale réalisée dans la section précédente chacune des variables apparaît dans la fonction d'hypothèse, éventuellement à plusieurs reprises à des puissances différentes.

[JAMES 2015f] James G., Witten D., Hastie T., Tibshirani R., *An Introduction to Statistical Learning*, 2e édition, New York : Springer, 2015, 61-71 p. (3.2 Multiple Linear Regression)

Prenons l'exemple de l'évaluation du prix d'une habitation sur base de ses données suivantes.

Superficie de la maison (en m^2)	Nombre de chambres	Nombre de salle de bain	Age de la construction	Superficie du terrain (en m^2)	Prix de vente
120	3	1	110	422	165000
165	3	1	130	687	145000
275	5	2	31	1680	290000
98	2	1	82	931	140000
…					
144	4	2	67	2568	192000

Données fictives de caractéristiques de maisons permettant l'évaluation de leur prix.

Dans ce cas, l'ensemble des données d'entrées sont arrangées en matrice X et en vecteur y comme suit :

$$X = \begin{pmatrix} 1 & 120 & 3 & 1 & 110 & 422 \\ 1 & 165 & 3 & 1 & 130 & 687 \\ 1 & 275 & 5 & 2 & 31 & 1680 \\ 1 & 98 & 2 & 1 & 82 & 931 \\ 1 & \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & 144 & 4 & 2 & 67 & 2568 \end{pmatrix} \qquad y = \begin{pmatrix} 165000 \\ 145000 \\ 290000 \\ 140000 \\ \cdots \\ 192000 \end{pmatrix}$$

On remarque qu'un vecteur colonne composé exclusivement de 1 est ajouté comme première colonne de la matrice X. C'est ce vecteur qui sera multiplié par θ_0, le premier multiplicateur du vecteur θ, pouvant déterminer le décalage indépendant des variables composant les données d'entrées.

La fonction d'hypothèse est construite pour tenir compte de l'ensemble des attributs d'une donnée d'entrée et en calculera le multiplicateur idéal pour prédire le prix de vente en minimisant la fonction d'estimation. En ne faisant apparaître qu'à une seule reprise chacune des variables de l'exemple précédent, l'hypothèse pourrait être construite comme suit :

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4 + \theta_5 x_5$$

où $x_0, x_1, \ldots$ représentent les vecteurs colonne de la matrice X

Dans ce cas, la technique du gradient (ou l'un de ses substituts) s'applique également agissant sur chaque paramètre simultanément déterminant à son terme le vecteur θ minimisant la fonction au regard des données d'entrée.

Dû au nombre de variables, la visualisation de l'ensemble de celles-ci au regard de la donnée à prédire est impossible. La représentation graphique ne pouvant représenter au mieux que trois dimensions. L'évaluation des différentes hypothèses doit donc être réalisée exclusivement par l'évaluation du coût avec le jeu de données de validation comme détaillé à la section III.B.1.a).

(1) Normalisation des variables (Normalization)

La diversité des paramètres ajoute une autre difficulté. Celle-ci rend l'opération de minimisation plus gourmande en ressources, de surcroît, lorsque les différentes variables sont établies dans des intervalles différents.

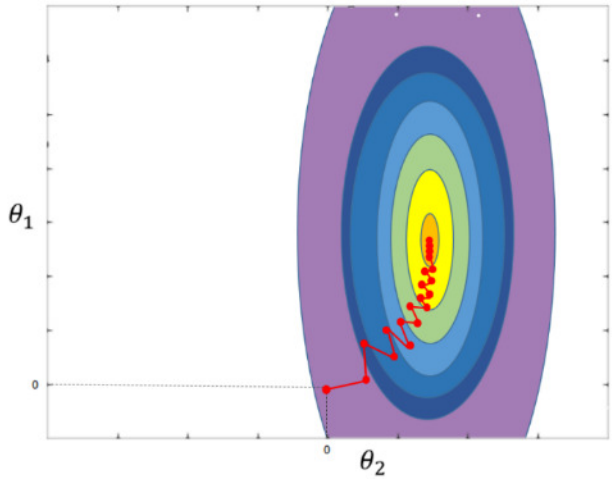
Comme indiqué dans le livre[SHALEV-SCHWARTZ 2014] « Understanding Machine Learning: From Theory to Algorithms » de Shai Shalev-Shwartz, professeur à l'université de Jerusalem et Shai Ben-David, professeur à l'université de Waterloo (Canada) et tous deux auteurs de nombreux travaux de recherches sur le *Machine learning*, il est important de normaliser les attributs des données d'entrée, c'est-à-dire de les transformer pour qu'ils soient représentés dans des intervalles limités et similaires pour chaque paramètre.

En l'absence de normalisation, l'algorithme du gradient, comme les autres algorithmes de minimisation, réalise un grand nombre de calculs intermédiaires supplémentaires car le pas réalisé à chacune des étapes se réalise dans une direction non-optimale. Cela s'explique par le fait que certains paramètres influent plus que de raison sur l'algorithme car leur poids est beaucoup plus grand.

L'image suivante illustre ce phénomène. Comme il est compliqué de représenter graphiquement les données au-delà de trois dimensions (au mieux, on en représente quatre –sur trois axes pour les trois premières dimensions, avec un jeu de couleurs pour rendre compte de la quatrième-), je représente ici une équation du type $h(x) = \theta_1 x_1 + \theta_2 x_2$ (où θ_0 serait nul) et ou les données

[SHALEV-SCHWARTZ 2014] Shalev-Shwartz Shai et Ben-David Shai, *Understanding Machine Learning: From Theory to Algorithms*, Cambridge University Press, 2014, 365-368 p. (*Feature Manipulation and Normalization*)

$x_1$et x_2 se situent dans des intervalles fortement différents car aucune normalisation n'aurait eu lieu.

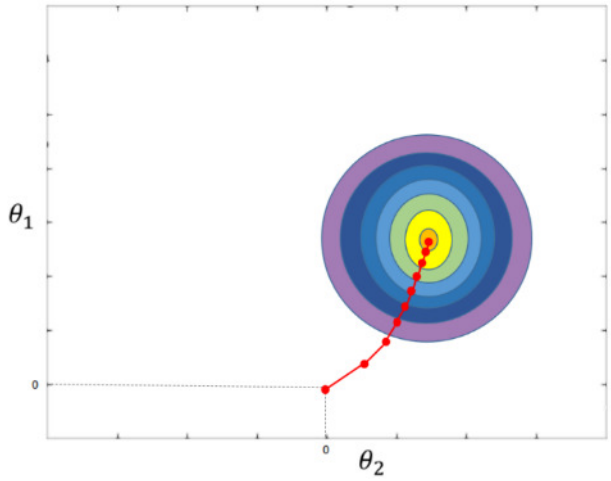


Minimisation sans normalisation, 1. [image inspirée d'une esquisse du cours en ligne de Coursera.org (Introduction to Machine Learning), 2015]

Représenté en trois dimensions (θ_1, θ_2 et $J(\theta)$ représenté ici par le jeu de couleurs allant du mauve, les valeurs élevées, au jaune, les valeurs minimales), on verrait l'algorithme du gradient réaliser un très grand nombre de pas en zigzagant comme représenté ici en rouge.

Pour éviter ce phénomène et rendre la minimisation plus rapide, on conseille, donc, de normaliser chacune des données d'entrées pour que celles-ci tiennent dans des intervalles similaires et réduits (*Features scaling*). On conseille encore parfois de s'assurer que leur moyenne soit proche de zéro, la normalisation par la moyenne (*Mean normalization*).

Pratiquement, avec des paramètres d'entrée normalisés, l'algorithme aboutit à un résultat équivalent en empruntant un chemin plus direct vers le minimum, en réalisant donc un nombre de pas réduit.



Minimisation avec normalisation, 1. [image inspirée d'une esquisse du cours en ligne de Coursera.org (Introduction to Machine Learning), 2015]

Une des méthodes proposées est de soustraire de la valeur d'un attribut de l'ensemble des données par leur valeur moyenne puis de la diviser par l'écart entre la valeur minimale et maximale.

Si nous reprenons l'exemple de l'estimation du coût d'une habitation ci-dessus, on peut calculer que la taille moyenne d'une maison, reprise dans nos données d'entrée comme x_1, est de 134 m². La différence la plus grande mesurée entre deux maisons est, quant à elle, de 177 m², on modifie alors toutes les données du vecteur x_1 comme suit :

$$x_1 := \frac{x_1 - 134}{177}$$

De manière analogue, on normalise les autres attributs comme suit :

$$x_2 := \frac{x_2 - 3{,}4}{3} \qquad x_3 := \frac{x_3 - 1{,}4}{1}$$

$$x_4 := \frac{x_4 - 84}{99} \qquad x_5 := \frac{x_5 - 1257{,}6}{2146}$$

On obtient alors la matrice X suivante :

$$X = \begin{pmatrix} 1 & 120 & 3 & 1 & 110 & 422 \\ 1 & 165 & 3 & 1 & 130 & 687 \\ 1 & 275 & 5 & 2 & 31 & 1680 \\ 1 & 98 & 2 & 1 & 82 & 931 \\ 1 & \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & 144 & 4 & 2 & 67 & 2568 \end{pmatrix}$$

Matrice X initiale

$$X = \begin{pmatrix} 1 & -0{,}228 & -0{,}133 & -0{,}4 & 0{,}263 & -0{,}389 \\ 1 & 0{,}026 & -0{,}133 & -0{,}4 & 0{,}465 & -0{,}266 \\ 1 & 0{,}647 & 0{,}533 & 0{,}6 & -0{,}535 & 0{,}197 \\ 1 & -0{,}352 & -0{,}467 & -0{,}4 & -0{,}02 & -0{,}15 \\ 1 & \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & -0{,}093 & 0{,}2 & 0.6 & -0{,}171 & 0{,}611 \end{pmatrix}$$

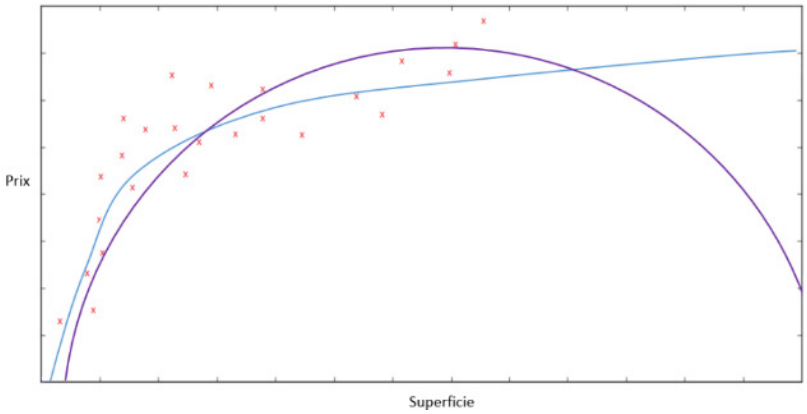
Matrice X normalisée

(2) Opérations différentiées dans l'hypothèse

Actuellement, la forme la plus complexe de la fonction hypothèse étudiée dans ce document en la forme polynomiale. Rien n'interdit pourtant de la complexifier lorsqu'intuitivement, on pressent un rapport mathématique particulier entre une variable et son résultat.

Si par exemple, nous constatons que le prix d'une maison augmente continuellement plus sa superficie grandit, il est plus judicieux d'opérer une racine carrée sur ce paramètre qu'une mise en carré. Le carré de la superficie engendre, en effet, un fléchissement décroissant pour des valeurs élevées.

$h(x) = \theta_0+\theta_1 x + \theta_2 x^2$ $h(x) = \theta_0+\theta_1 x + \theta_2\sqrt{x}$



Choix des fonctions, 1. [image inspirée d'une esquisse du cours en ligne de Coursera.org (Introduction to Machine Learning), 2015]

Il faut garder à l'esprit que la normalisation des paramètres d'entrée est sensible à l'opération qui est appliquée aux variables. Si nous utilisons, par exemple, le modèle suivant $h(x) = \theta_0+\theta_1 x + \theta_2\sqrt{x}$ avec l'hypothèse $h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2$. Sans normalisation par la moyenne (pouvant engendrer le calcul de la racine carré d'un nombre négatif), la matrice X sera constituée comme suit :

Le vecteur colonne x_0 est toujours composé de 1 et échappe à toute normalisation.

Le vecteur colonne x_1 prendra la valeur x et sera normalisé comme suit :

$$\forall i \in [0; m-1], x_1^{(i)} := \frac{x_1^{(i)}}{maximum(x_1) - minimum(x_1)}$$

Le vecteur colonne x_2 prendra la valeur x et sera normalisé comme suit:

$$\forall i \in [0; m-1], x_2^{(i)} := \frac{\sqrt{x_2^{(i)}}}{\sqrt{maximum(x_2) - minimum(x_2)}}$$

où $x^{(i)}$ représente le i[ième] enregistrement de x.

c) Que faire pour améliorer les résultats d'un algorithme ?

Si, malgré nos efforts, la régression réalisée ne permettait pas de réaliser des prédictions réalistes, il existe un certain nombre de mesures pouvant encore être entreprises.

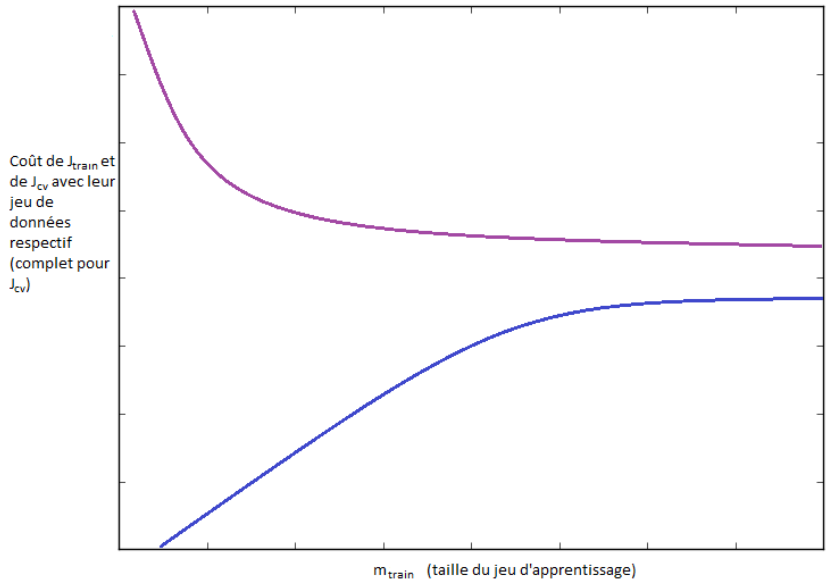
(1) Obtenir plus de données

La première mesure à prendre est de s'interroger sur le fait que notre jeu de données soit suffisant et si l'utilisation d'enregistrements complémentaires n'améliorerait pas la décision de notre modèle.

Pour en juger, il est utile de regarder l'évolution du coût en augmentant petit à petit le nombre d'exemples du jeu d'entrainement.

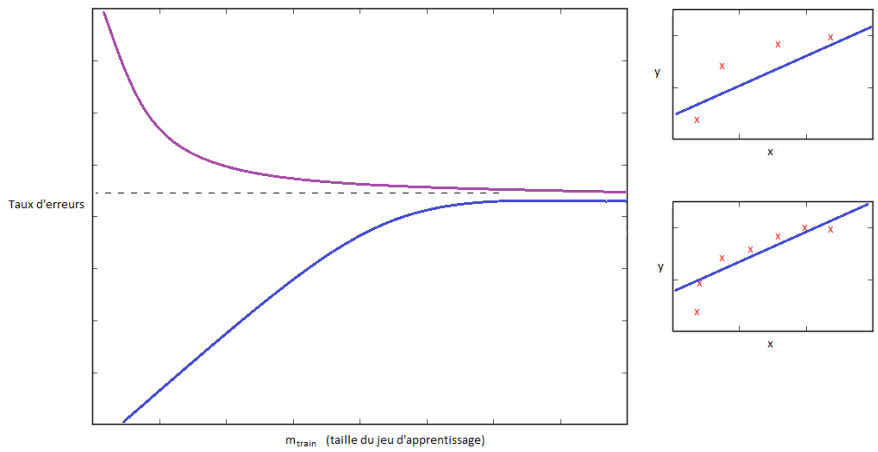
Plus le nombre d'exemples du jeu d'apprentissage (pas celui du jeu de validation) augmente plus le coût global de l'apprentissage devrait augmenter. Inversement, le coût global de la phase de validation croisée devrait diminuer.

$$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta\left(x_{train}^{(i)}\right) - y_{train}^{(i)})^2 \qquad J_{cv}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta\left(x_{cv}^{(i)}\right) - y_{cv}^{(i)})^2$$



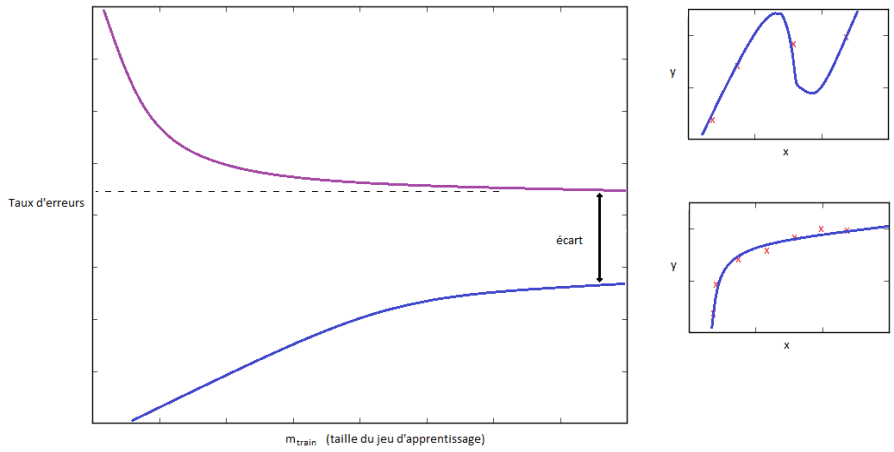
Evolution du coût en fonction du volume de données, 1. [image inspirée d'une esquisse du cours en ligne de Coursera.org (Introduction to Machine Learning), 2015]

Si le taux d'erreurs est très élevé et que la différence entre le coût de $J_{train}(\theta)$ et $J_{cv}(\theta)$ est faible comme représenté ci-dessous, c'est que la fonction souffre de biais important (high bias). C'est qu'il généralise excessivement. Dans ce cas, la collecte de nouvelles données est superflue ; celle-ci n'améliorera que très peu les prédictions. Il est donc plus intéressant de travailler sur la fonction d'hypothèse.



Evolution du coût en fonction du volume de données, 2. [image inspirée d'une esquisse du cours en ligne de Coursera.org (Introduction to Machine Learning), 2015]

Si, à l'inverse, le taux d'erreur devait être faible par le calcul de $J_{train}(\theta)$ et mais important avec celui de $J_{cv}(\theta)$, il est probable que l'ajout de données aux jeux existants puisse aider. C'est que le modèle souffre de variance importante (high variance).



Evolution du coût en fonction du volume de données, 3. [image inspirée d'une esquisse du cours en ligne de Coursera.org (Introduction to Machine Learning), 2015]

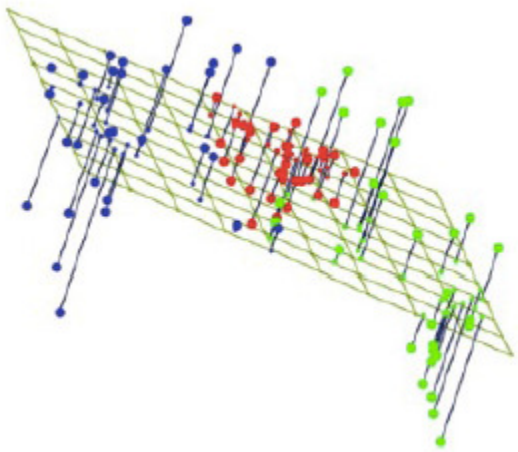
(2) Utiliser moins de variables à la fois

Certaines des caractéristiques utilisées ne sont peut-être pas représentatives du domaine d'application ou sont beaucoup trop volatiles (éventuellement peu précises) pour influer de manière significative sur le résultat. Lorsqu'un modèle offre une précision insuffisante, il peut être utile d'ôter l'un ou l'autres des paramètres pour s'assurer que celles-ci ne génèrent pas de bruit. La diminution du nombre de variables permettra également d'améliorer la rapidité de l'apprentissage.

Plusieurs techniques existent à cette fin. Outre le fait que cette méthode soit très utile pour visualiser les résultats d'une classification, la méthode PCA[NG 2015c] (*Principal component analysis*) permet, par exemple, la compression des *features* pour en réduire le nombre en projetant celles-ci dans un espace de dimensions inférieur.

[NG 2015c] Ng Andrew, *Part XI Principal components analysis*, http://cs229.stanford.edu/notes/cs229-notes10.pdf, 2015, consulté le 20 avril

L'image suivante illustre par exemple la réduction de données tridimensionnelles sur un plan en deux dimensions. Pour éviter une projection excessivement néfaste à la qualité des données, le plan choisi est celui dont le coût de l'aplatissement est le plus faible.



Source : Understanding Dimensionality Reduction- Principal Component Analysis And Singular Value Decomposition. http://www.bigdataexaminer.com/wp-content/uploads/2014/12/17.png

La technique Lasso[HASTIE 2015] (*Least Absolute Shrinkage and Selection Operator*, pouvant se traduire par « la moindre réduction et l'opérateur de sélection ») est aussi utile pour réduire le nombre de variables utilisées par le modèle. Cette méthode réduit en effet le poids de certaines variables à 0 si celles si n'améliorent pas significativement, selon un seuil déterminable, le modèle.

(3) Tenter d'utiliser des fonctions spécifiques.

Dans certains domaines d'application, certaines caractéristiques se renforcent ou se neutralisent. D'autres n'ont d'intérêt qu'en présence d'une autre valeur élevée ou basse sur une autre caractéristique. La multiplication ou la division de caractéristiques entre elles permettrait de matérialiser leur dépendance mutuelle.

De manière complémentaire, tel qu'il l'a déjà été abordé précédemment, la recherche de fonctions spécifiques telles que la puissance, la racine, le logarithme, etc. peuvent être

[HASTIE 2015] Hastie T., Tibshirani R., Wainwright M., *The statistical Learning with Sparsity*, 1e édition, Chapman and Hall, 2015, 1-24 p. (1. Introduction, 2.The Lasso for Linear Models)

appliquées sur les variables. Les visualiser graphiquement une par une au préalable au regard du résultat aide à la prise de décision de l'opération appropriée.

(4) Introduire la régularisation

Comme le détaille le livre « *The Elements of Statistical Learning* »[HASTIE 2008] de Trevor Hastie, Robert Tibshirani et Jerome Friedman, la régularisation est le terme, adjoint à la fonction de coût, employé pour désigner la manière dont il est possible de nuancer l'importance de chaque multiplicateur dans la constitution de notre modèle durant la phase d'apprentissage. Cette régularisation consiste à l'ajout d'un terme dans la fonction de coût utilisée lors de cette phase. Ce terme influence le modèle qui sera issu de la phase d'apprentissage en prévenant l'« *overfitting* » en s'assurant que les valeurs du vecteur θ ne deviennent pas démesurées.

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m}\sum_{j=1}^{m}\theta_j^2$$

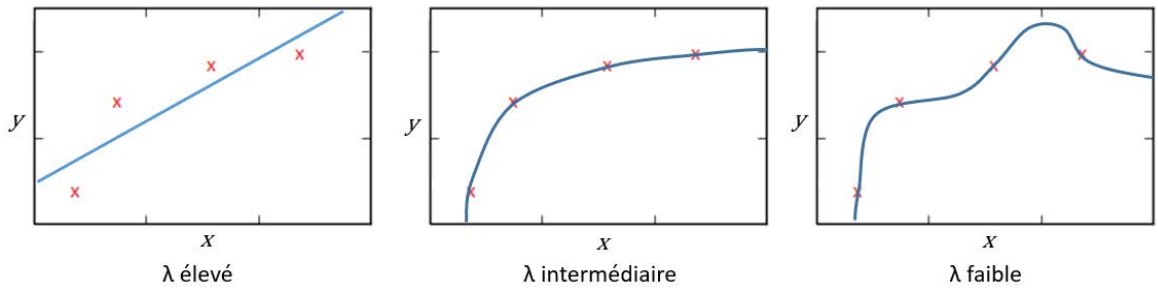
La régularisation se manifeste par une pénalisation sur le coût de toutes augmentations de chacune des métriques composant le vecteur θ lors de l'application d'une méthode d'optimalisation, telle que le gradient descendant.

En pratique, bien que la fonction d'hypothèse puisse être complexe, le paramètre de régularisation forcera une généralisation minimale. L'ajustement du paramètre est raisonnablement à réaliser par palier, par exemple, de 0 à 0.01, 0.02, 0.04, … jusqu'à 10.

Le facteur λ a la vocation d'être ajustable. En jouant sur ce paramètre, il devient possible de modifier la capacité de la fonction hypothèse à généraliser. Plus λ est important, plus celui-ci favorise le biais de la fonction de prédiction des données. Inversement, si celui-ci est nul ou très faible, il y a pas ou peu de régularisation.

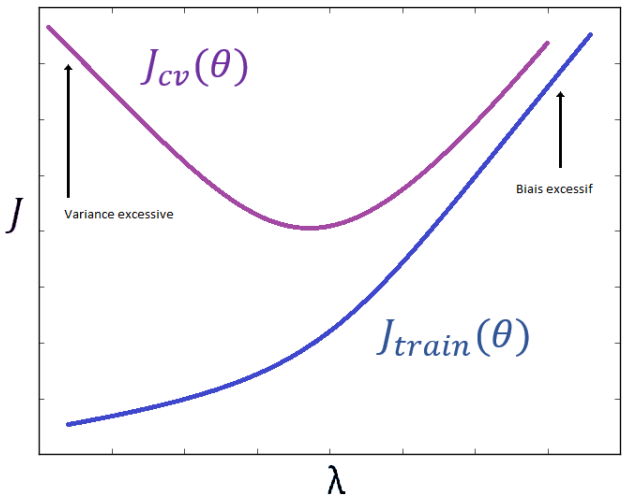
[HASTIE 2008] Hastie T., Tibshirani R. et Friedman J., *The Elements of Statistical Learning*, 2e édition, New York : Springer-Verlag, 2008, 37-38 p. (*Model Selection and the Bias–Variance Tradeoff*)

Avec une hypothèse polynomiale, il serait plausible de modifier les courbes comme suit :



Ajustement du paramètre λ, 1.

Si les résultats de notre hypothèse devaient manquer leur cible, on recommande donc de tenter de réduire le paramètre de régularisation λ (pour forcer davantage de variance) ou de l'augmenter le (pour engendrer davantage de biais) afin de trouver un meilleur minimum sur le jeu de validation croisée.



Ajustement du paramètre λ et impact sur le coût, 1. (2016). [traduction et adaptation des notations de la figure 2.11. Test and training error as a function of model complexity, Trevor Hastie, Robert Tibshirani et Jerome Friedman, *The Elements of Statistical Learning*, 2e édition, New York : Springer-Verlag, 2008, 38 p. (*Model Selection and the Bias–Variance Tradeoff)]*

(5) Obtenir d'autres features que celles déjà utilisées

Si aucune des actions précédentes n'a montré son efficacité, il est probable que le domaine d'application ne soit pas régi par les données que nous avons collectées. Celles-ci sont soit fortement imprécises, incomplètes ou non représentatives. Dans ce cas, seule la collecte de

nouvelles caractéristiques aux données en notre possession peut permettre d'améliorer nos résultats.

3. Régression logistique

Jusqu'à présent, le présent mémoire n'a fait état que de méthodes permettant de déterminer des prédictions quantitatives. Le *Machine learning* comprend également un ensemble de méthodes permettant de déterminer des attributs qualitatifs d'une donnée, les algorithmes de classification[JAMES 2015g].

La régression logistique est une de ces méthodes d'apprentissage automatique permettant la réalisation de classification ayant pour objectif de prédire l'appartenance d'une donnée à des ensembles identifiés et donc une caractéristique qualitative.

Comme dans la cadre des méthodes précédentes, l'apprentissage tentera de déterminer la (ou les) frontière la plus appropriée pour tenter de séparer les données au départ d'exemples dont la caractéristique est connue. De nombreuses applications de ces algorithmes existent. Elles sont, par exemple, utilisées pour prédire ensuite le caractère frauduleux d'une transaction financière ou encore la classification des emails comme étant du spam ou non, tenter de prédire la couleur des yeux sur base de données ADN.

a) Représentation de l'hypothèse

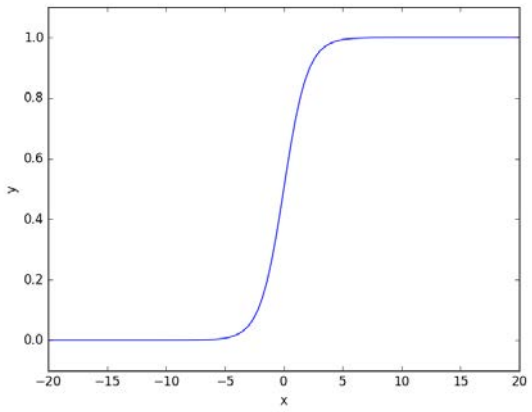
Pour réaliser la fonction hypothèse adéquate pour une classification, on applique la fonction « sigmoïde », également appelée fonction logistique, sur l'hypothèse adaptée aux régressions.

Pour une régression : $h_\theta(x) = \theta^T x$	Pour une classification : $h_\theta(x) = g(\theta^T x)$

Où $g()$ représente la fonction sigmoïde $g(x) = \frac{1}{1+e^{-x}}$ où e est le nombre d'Euler (2,71828).

Cette fonction à la particularité de retourner presqu'exclusivement des valeurs très proches de 0 ou de 1 et donc de retourner un résultat quasiment binaire autour d'une frontière de décision.

[JAMES 2015g] James G., Witten D., Hastie T., Tibshirani R., *An Introduction to Statistical Learning*, 2e édition, New York : Springer, 2015, 127-150 p. (4. Classification)

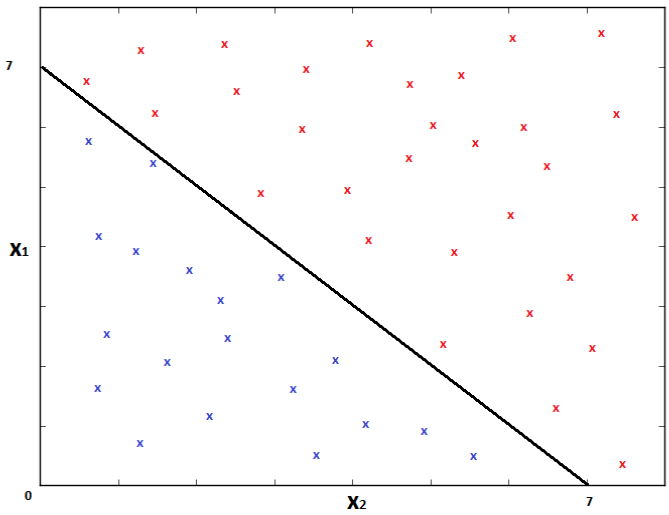


Fonction sigmoïde, 1.

b) La fixation de la frontière de décision (Decision Boundary)

La frontière de décision est doit être déterminée au départ d'exemples connus en les segmentant en deux ensembles et en en épousant la frontière la plus cohérente. Celle-ci peut donc prendre des formes simples, telles qu'une droite ou une courbe, ou des formes beaucoup plus complexes.

Prenons l'exemple suivant dans lequel on perçoit aisément une frontière linéaire claire.

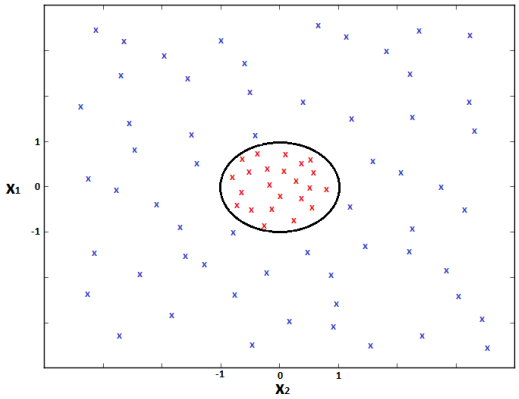


Fixation d'une frontière de décision, 1.

Si l'algorithme doit retourner 1 pour identifier la donnée comme positive si $x_1 + x_2 > 7$, à défaut elle doit retourner 0, la fonction hypothèse pourrait alors s'exprimer comme suit :

$$h_\theta(x) = -7 + x_1 + x_2$$

Pour représenter des frontières plus complexes, comme un ensemble fermé par exemple, il est recommandé de déterminer l'hypothèse au moyen d'une équation polynomiale.

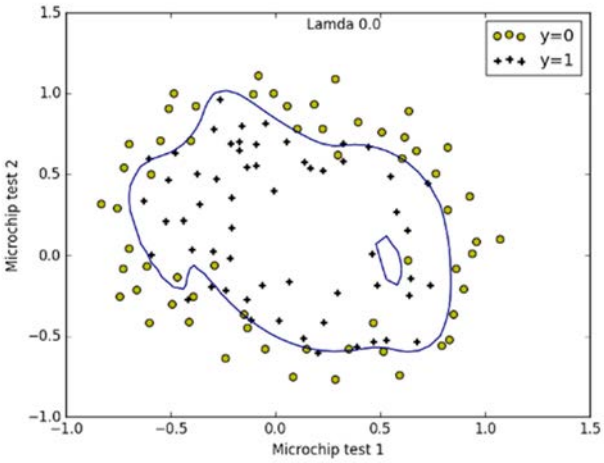


Fixation d'une frontière de décision, 2.

Dans cet exemple, la frontière entre des données représentées ici en rouge et bleu peut être modélisée par l'équation hypothèse suivante $h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$

où le vecteur $\theta = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$

Des formes encore plus complexes peuvent être obtenues en augmentant le degré polynomial comme illustré ci-dessous. Théoriquement, plus on complexifie la fonction d'hypothèse, plus la frontière peut être irrégulière.



Fixation d'une frontière de décision, 1. [données mises à disposition sur la plateforme Coursera.org (Introduction to Machine learning), 2015]

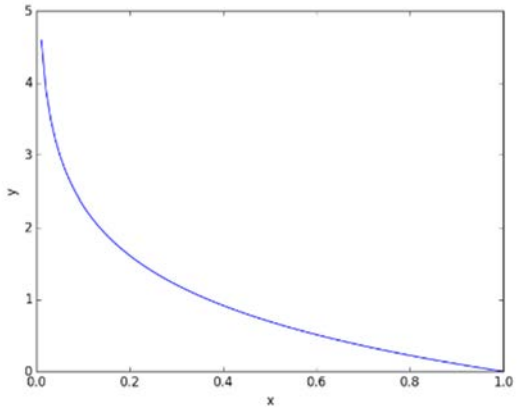
Le calcul du vecteur θ est alors calculé de manière analogue aux méthodes de régression linéaire et non linéaire par le biais d'une fonction de minimalisation mais avec une fonction de coût adaptée aux régressions logistiques (la présence de la fonction sigmoïde pourrait rendre la fonction de coût vue précédemment non convexe, c'est-à-dire qu'elle pourrait comporter plusieurs minimums locaux).

La fonction de coût à utiliser est adaptée au souhait de classification. Par sa structure, elle comptabilise le nombre d'erreurs de classification :

$$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))]$$

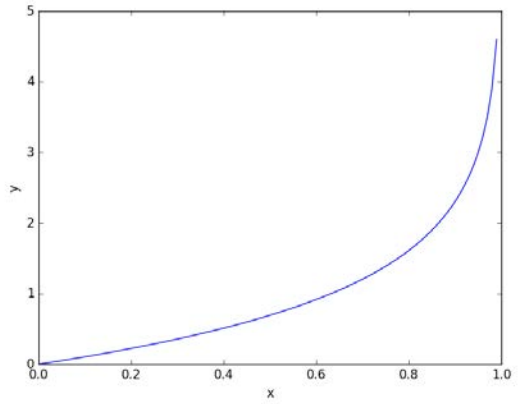
Si on considère que le terme y ne peut prendre que les valeurs 0 ou 1 (suivant que le résultat soit positif ou négatif), l'un des deux termes ($y^{(i)} \log h_\theta(x^{(i)})$ et $(1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))$ est systématiquement annulé. L'autre terme prend, par contre, une valeur très grande en cas d'erreur de prédiction et très faible pour toutes valeurs dont la prédiction s'avère correcte. Le coût augmentera donc fortement à chaque classification erronée.

Si $y = 1$, $-\log(h_{\theta(x)})$ dessine, en effet, la courbe suivante, toutes données évaluées erronément à une valeur proche de 0 pénalise très fortement le coût global de notre hypothèse :



Fonction $-log(h_{\theta(x)})$*, 1.*

Si $y = 0$, $-\log(1 - h_{\theta(x)})$ dessine la courbe suivante, toutes données évaluées erronément à une valeur proche de 0 pénalise très fortement le coût global de notre hypothèse :



Fonction $-log(1 - h_{\theta(x)})$*, 1.*

Une fois l'hypothèse entrainée avec les données du jeu d'entrainement, les prédictions peuvent être réalisées au moyen de la fonction sigmoïde :

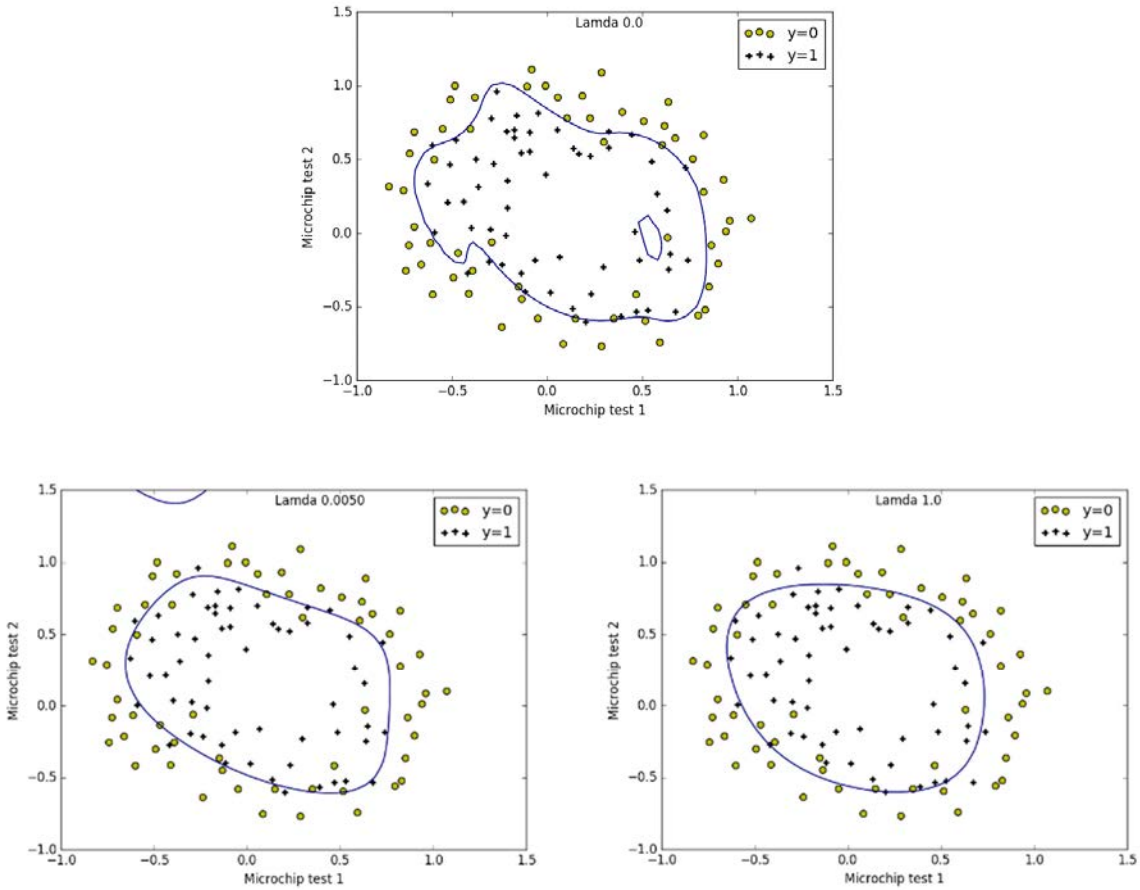
$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

où $\theta^T x$ est la fonction hypothèse.

c) Régularisation

Comme dans les régularisations linéaire ou non linéaire, la régularisation peut utile pour résoudre des problèmes de variance excessive. En ajoutant le terme de régularisation et en l'augmentant par pas, il est possible de renforcer le biais d'une régression logistique.

$$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))] + \frac{\lambda}{2m}\sum_{j=1}^{n} \theta_j^2$$



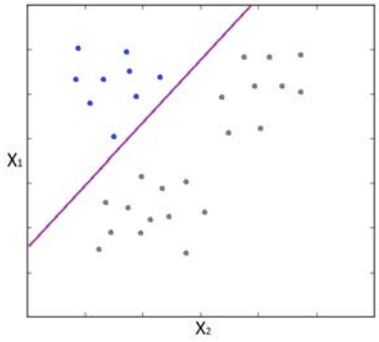
Fixation d'une frontière de décision, 2, 3, 4. [données mises à disposition sur la plateforme Coursera.org (Introduction to Machine learning), 2015]

d) Ensembles multiples (un contre tous)

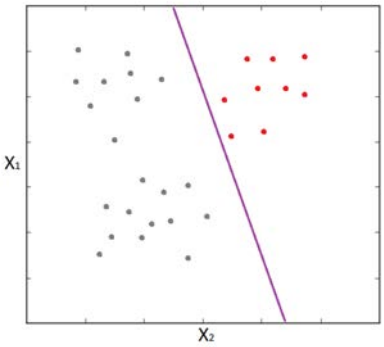
Les méthodes de classification ne sont pas exclusivement réservées à une classification « binaire » c'est-à-dire à la fixation de la frontière entre deux ensembles. On pourrait successivement évaluer l'appartenance d'une nouvelle donnée à un ensemble ou à tous les autres. Si celle-ci n'était pas membre du premier ensemble, on recommencerait alors l'opération sur l'une des autres catégories.

Trois ensembles pourraient, par exemple, être distingués au moyen de trois hypothèses différentes :

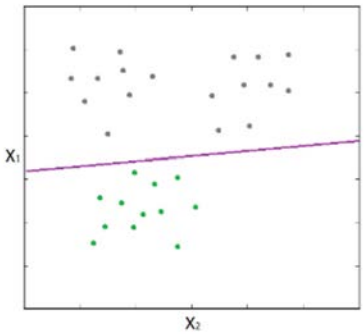
Hypothèse 1



Hypothèse 2



Hypothèse 3



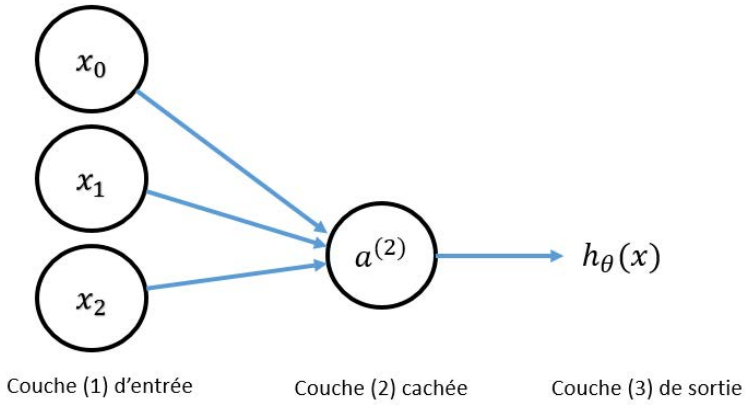
Fixation d'une frontière de décision, 5, 6, 7. [image inspirée d'une esquisse du cours en ligne de Coursera.org (Introduction to Machine Learning), 2015]

Dans ce cas, une même donnée pourrait appartenir à plusieurs ensembles mais la fixation d'autres frontières de décision que celle-ci pourrait rendre ces ensembles exclusifs.

4. Réseaux neuronaux (Neural Networks)

Les réseaux neuronaux tiennent leur nom du fonctionnement des neurones du cerveau dont ils s'inspirent et dont ils empruntent la terminologie. En biologie, ces neurones sont des cellules nerveuses qui réagissent ou non à des impulsions électriques particulières (nommées stimuli) par d'autres impulsions à l'attention d'autres neurones. De manière analogue, les réseaux neuronaux du domaine de l'apprentissage automatique sont dotés d'unités possédant des entrées et sorties. Les unités sont parfois appelées « cellules » et leurs connexions « synapses ».

La décision de sortie d'une unité est le résultat d'un apprentissage qui prend la forme d'une fonction mathématique dont les multiplicateurs ont été adaptés pour minimiser le coût. Cette fonction prend la forme d'une fonction sigmoïde appliquée sur une équation polynomiale $\theta^T x$.



Représentation schématique d'un réseau neuronal, 1. [image inspirée d'une esquisse du cours en ligne de Coursera.org (Introduction to Machine Learning), 2015]

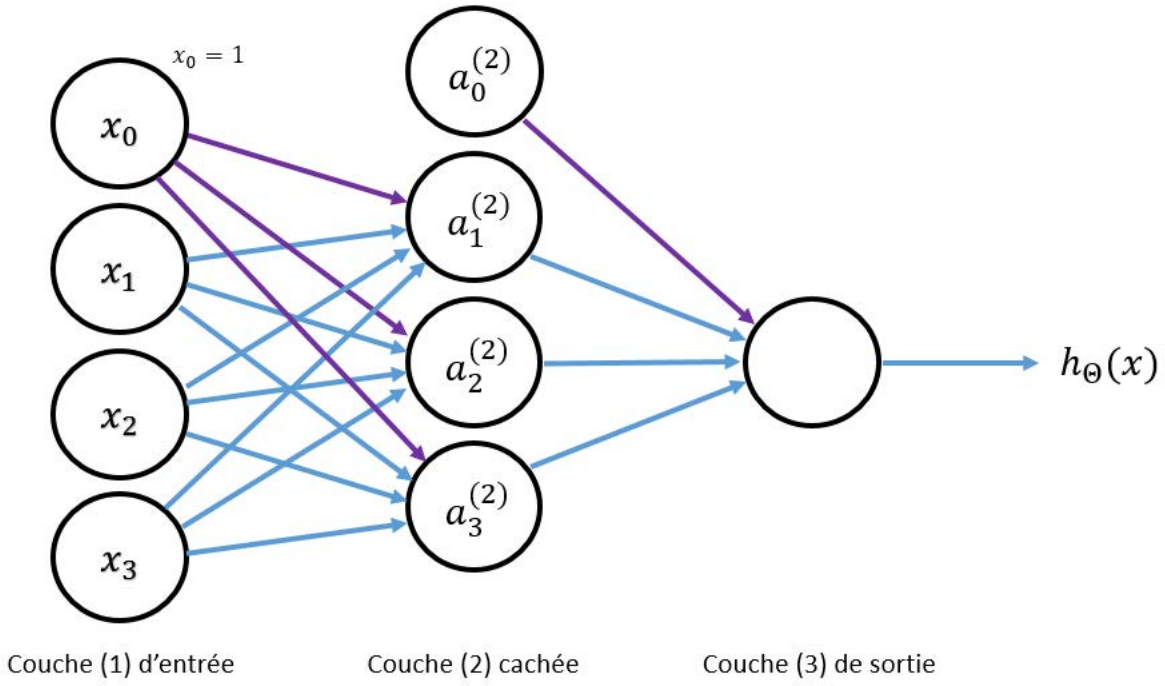
Si nous focalisons sur une unité, notée ici $a^{(2)}$ (où 2 désigne la couche), l'hypothèse $h_\theta(x)$ sera déterminée par la minimisation de la fonction de coût par l'ajustement des multiplicateurs θ tel que

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

Où, dans notre exemple, $\theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2$

L'hypothèse $h_\theta(x)$ retournera donc une matrice essentiellement composée de valeurs légèrement supérieures à zéro ou légèrement inférieures à 1.

La force de ce type des réseaux neuronaux résident dans leur caractéristique à interconnecter un grand nombre d'unités. Ces unités composent alors un réseau organisé en plusieurs couches. La première est appelée la couche d'entrée, la dernière la couche de sortie et toutes les autres (au minimum une comme dans l'exemple ci-dessous) sont dites cachées.



Représentation schématique d'un réseau neuronal, 2. [image inspirée d'une esquisse du cours en ligne de Coursera.org (Introduction to Machine Learning), 2015]

La couche d'entrée recevra les données d'entrée. Il existe donc autant d'unités que de variables auxquelles on ajoute x_0, la variable issue du premier vecteur colonne (composé exclusivement de 1) de la matrice X qui contient nos données d'entrées. De manière analogue, une unité « Bios » est ajoutée comme premier élément de chaque autre couche.

La couche de sortie retourne le résultat.

a) Propagation vers l'avant (forward propagation)

Les couches cachées incarnent le centre de décision. Chaque unité, appelée unité d'activation (notée a), est un centre de calcul qui retournera une décision, de la forme d'une matrice, aux unités suivantes.

$$a_1^{(2)} = g\left(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3\right)$$

$$a_2^{(2)} = g\left(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3\right)$$

$$a_3^{(2)} = g\left(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3\right)$$

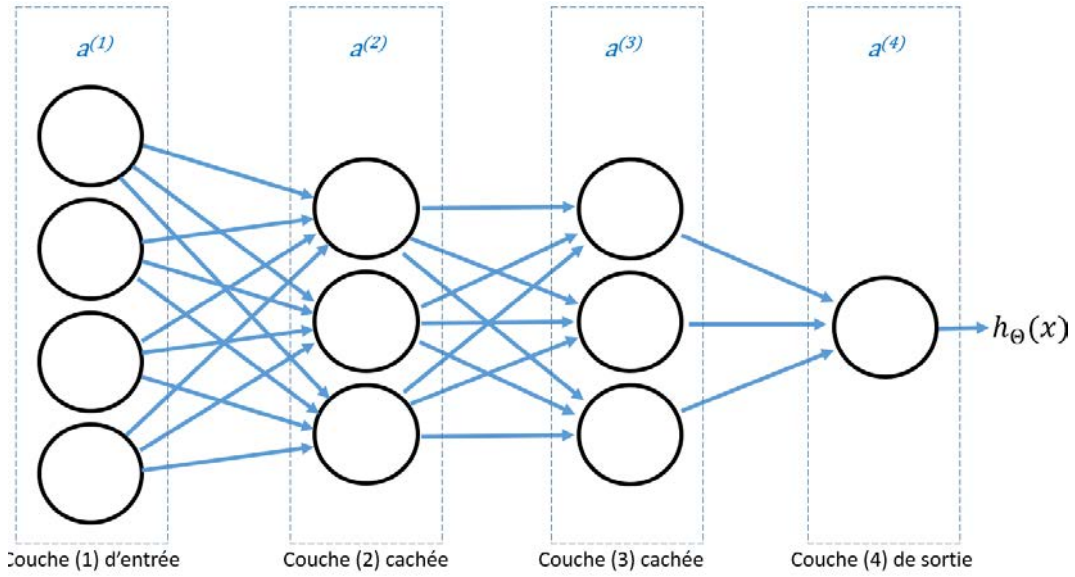
$$h_\Theta(x) = g\left(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)}\right)$$

où $\Theta_{mo}^{(n)}$représente le o^ième^ élément de la matrice de poids d'entrée calculée pour la m^ième^ unité de la n^ième^ couche.

Il est à remarquer que, dans notre schéma précédant, $a^{(2)}$ est conçu par la constitution d'un nouveau vecteur intégrant le résultat de toutes les unités de la couche (ici unique) cachée. Il s'agit donc d'un vecteur de matrice, un vecteur en trois dimensions.

$$a^{(2)} = \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{bmatrix}$$

Lorsqu'un réseau neuronal est composé de plusieurs couches cachées, chacune des couches prend en entrée les résultats de la couche précédente comme illustré ci-après :



Représentation schématique d'un réseau neuronal, 2. [image inspirée d'une esquisse du cours en ligne de Coursera.org (Introduction to Machine Learning), 2015]

La propagation se réalise alors comme suit :

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g(z^{(2)})$$

$$z^{(3)} = \Theta^{(2)} a^{(2)}$$

$$a^{(3)} = g(z^{(3)})$$

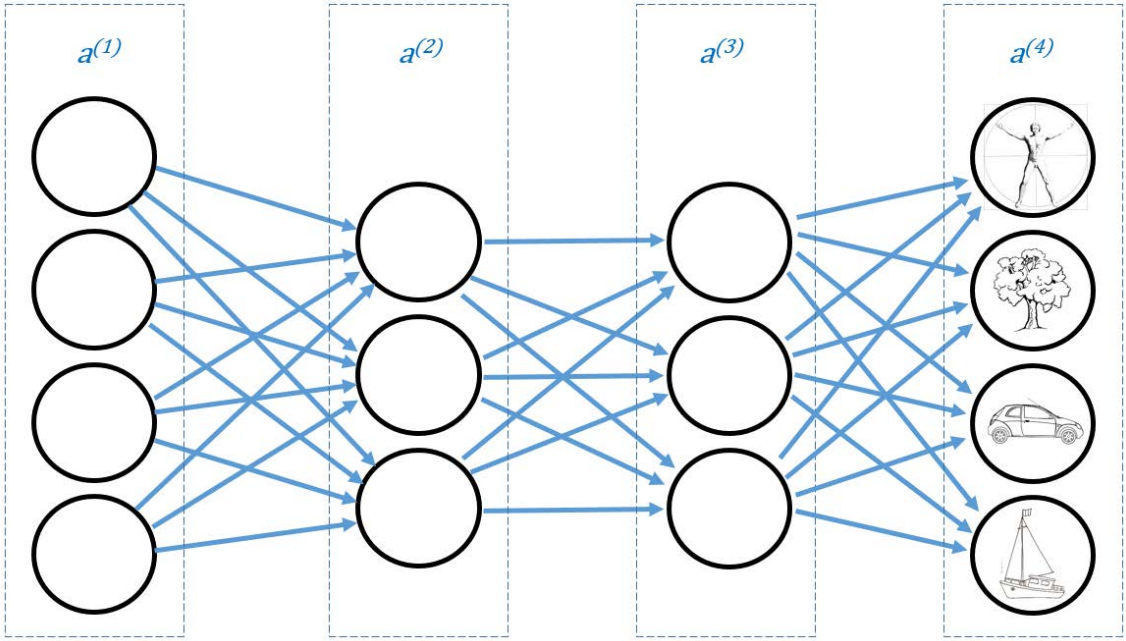
$$z^{(4)} = \Theta^{(3)} a^{(3)}$$

$$a^{(4)} = g\left(z^{(4)}\right) = h_\Theta(x)$$

b) Classification avec de multiples catégories

La couche de sortie peut comporter plusieurs unités. Cela permet de distinguer plusieurs catégories identifiables avec un même réseau neuronal.

On pourrait par exemple entrainer un réseau neuronal à distinguer les hommes, des arbres, des voitures et des bateaux.



Représentation schématique d'un réseau neuronal, 3. [image inspirée d'une esquisse du cours en ligne de Coursera.org (Introduction to Machine Learning), 2015]

Le résultat d'une hypothèse consiste alors dans un vecteur vertical composé de valeur extrêmement proche de zéro et d'un un représentant la catégorie identifiée.

$h_\Theta(x) \approx \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$ représentant un arbre dans l'exemple ci-dessus.

c) Coût

Le calcul du coût de la fonction hypothèse s'effectue de manière analogue aux méthodes précédentes à l'exception près que le vecteur Θ est initialisé aléatoirement. Ces paramètres de multiplication seront modifiés successivement lors de l'optimisation de la fonction hypothèse.

La fonction de coût ressemble à celle utilisée pour les algorithmes de classification. Elle a pour objectif de comptabiliser les erreurs de classification.

$$J(\Theta) = -\frac{1}{m}\left(\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(h_\Theta(x^{(i)})\right)_k + (1 - y_k^{(i)}) \log\left(1 - \left(h_\Theta(x^{(i)})\right)\right)_k\right) + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l}(\Theta_{ji}^{(l)})^2$$

où k représente le $k^{ième}$ élément du vecteur de résultat

d) Algorithme de rétropropagation du gradient (Back propagation algorithm)

Afin de minimiser la fonction hypothèse, il est nécessaire de dériver la fonction de coût au regard de l'ajustement des poids associés à chaque unité. Chaque ajustement doit permettre de diminuer le coût à la recherche du minimum.

Cette minimisation demande le calcul du coût de la fonction d'hypothèse $J(\Theta)$ avec la matrice Θ en l'état actuel et sa dérivée $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$.

Le calcul du coût $J(\Theta)$ est calculé au départ de la fonction de coût reprise en section c).

La dérivée de la fonction de coût s'obtient, pour sa part, au moyen de l'algorithme de rétropropagation. Cet algorithme tient son nom du fait qu'il calcule la quantité d'erreurs $\delta_j^{(l)}$(où l représente la couche et j représente l'unité) en partant de la couche de sortie, puis en calculant successivement les précédentes jusqu'à la couche 2.

Concrètement, l'erreur de chaque unité est calculée pour chaque donnée d'entrée et comparée au résultat attendu (à vrai dire celui-ci lui est soustrait).

$$\delta_j^{(l)} = a_j^{(l)} - y_j$$

De manière vectorisée, où un vecteur représente une couche, la rétropropagation s'effectue comme suit :

$$\delta^{(l-1)} = (\Theta^{(l-1)})^T \delta^l .* g'(z^{(l-1)})$$

où $g'(z^{(l-1)})$ est la dérivée de la sigmoïde de la fonction d'activation avec les multiplicateurs de $z^{(l-1)}$. $g'(z^{(l-1)})$ est mathématiquement équivalent à $a^{(l-1)} .* (1 - a^{(l-1)})$.

Cette méthode possède l'avantage de ne pas devoir dériver $J(\Theta)$ de manière classique ce qui s'avèrerait particulièrement lent et coûteux en ressources, la dérivée $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$ pouvant être déterminée au départ du calcul de $a_j^{(l)} \delta_i^{(l+1)}$ si on ignore toute régularisation $(\lambda = 0)$[MEEDEN].

e) Entrainement

L'entrainement du réseau neuronal s'effectue par l'alternance de propagation vers l'avant et de rétropropagation.

[MEEDEN] Meeden Lisa, *Derivation of Backpropagation*, https://www.cs.swarthmore.edu/~meeden/cs81/f15/BackPropDeriv.pdf, date inconnue, consulté le 10 février 2016

Concrètement, après avoir initialisé l'accumulateur $\Delta_{ij}^{(l)}$ de manière aléatoire pour tout l, i, j (cela permet de s'assurer que le paramètre $\Theta_{ij}^{(l)}$ de deux unités d'un même couche ne soient pas modifiés de manière identique pendant l'algorithme – on parle de rupture de la symétrie)), on réalise l'opération suivante :

pour chaque donnée du jeu de données d'entrée :

1. On effectue la propagation vers l'avant. On assigne $a^{(i)} = x^{(i)}$ et on calcule successivement chaque couche $a^{(l)}$ jusqu'à la couche de sortie
2. On calcule le coût de notre hypothèse, c'est-à-dire qu'on compare le résultat avec le résultat des données d'entrée $y^{(i)}$et on calcule l'erreur de la couche de sortie $\delta^{(L)} = a^{(L)} - y^{(i)}$
3. On utilise la rétropropagation pour calculer l'erreur des couches précédentes jusqu'à la couche 2.
4. On modifie $\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

On introduit ensuite la régularisation sur toute la matrice à l'exception des unités bios.

$$D_{ij}^{(l)} := \frac{1}{m}\Delta_{ij}^{(l)} + \lambda\, \Theta_{ij}^{(l)} \text{ si } j \neq 0 \qquad\qquad D_{ij}^{(l)} := \frac{1}{m}\Delta_{ij}^{(l)} \text{ si } j = 0$$

La dérivée $D_{ij}^{(l)}$peut alors être utilisée avec une méthode d'optimisation telle que le gradient descendant car celle-ci est mathématiquement équivalente à $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$.

f) Vérification

Pour vérifier que l'algorithme d'optimisation fonctionne correctement, il est possible de simuler l'ajustement que devrait réaliser le gradient en simplifiant la dérivée de la fonction de coût au moyen d'une méthode non vectorisée (où on agit sur une seul variable à la fois).

On initialise le vecteur Θ (variabilisé theta ci-dessous) aux mêmes valeurs que les nombre aléatoires choisis pour l'entrainement. Ensuite, pour chaque variable de notre jeu de données

d'entrée (éventuellement réduit car beaucoup plus lent), on évalue si le coût diminue en diminuant ou augmentant le poids de chaque variable.

En python, cela pourrait se traduire comme suit :

```
for i in range (0,n):
        thetaPlus = theta
        thetaPlus(i) = thetaPlus(i) + EPSILON
        thetaMinus = theta
        thetaMinus(i) = thetaMinus(i) – EPSILON
        gradApprox(i) = (J(thetaPlus) – J(thetaMinus)) / (2 * EPSILON)
```

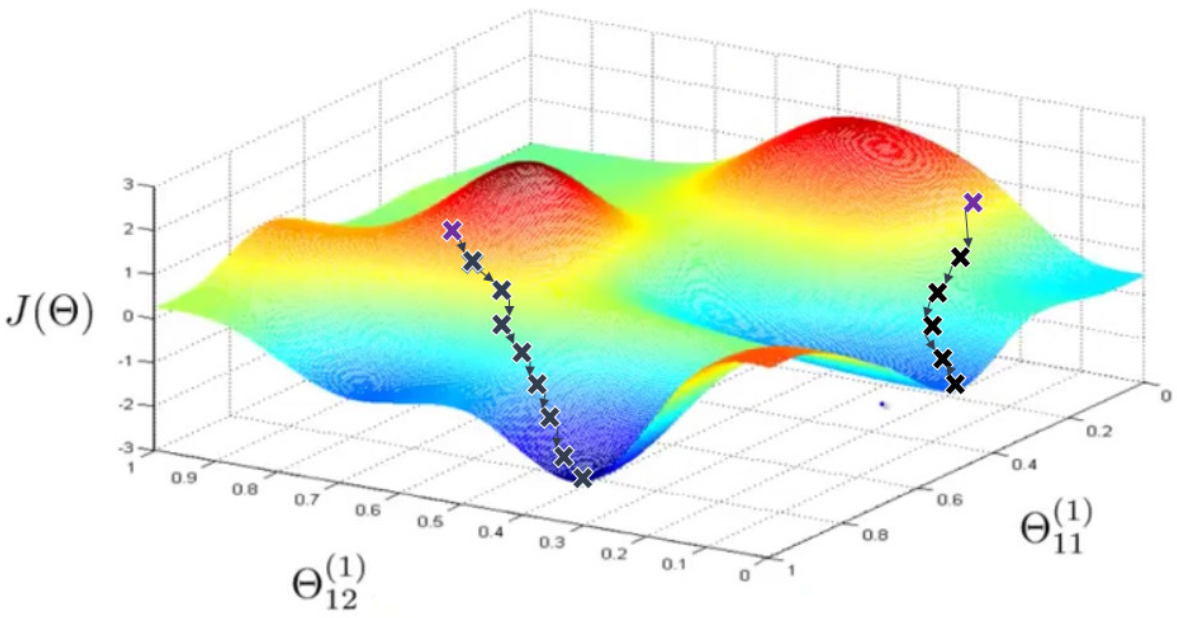
où la fonction J() retourne le coût.

L'approximation (ici sous forme du vecteur inclus dans la variable gradApprox) devrait approcher le vecteur Θ obtenu à l'issue de l'entrainement à chaque itération.

g) Validation

Après avoir joué une méthode d'optimisation sur notre hypothèse, nous pouvons évaluer sa précision. Comme lors des exercices d'apprentissage précédents, les phases de validation et de test permettent enfin de juger de la performance et du caractère biaisé ou de la variance éventuellement excessive de l'équation.

Il est possible que la fonction de coût ne soit pas convexe. Cela pourrait orienter l'algorithme à converger vers un minimum local non optimal. Dans ce cas, on recommande de lancer plusieurs fois l'optimisation avec une matrice Θ initialisée avec des valeurs aléatoires, permettant ainsi de multiplier les chances de trouver le minimum global comme l'illustre l'image-ci-dessous.



Source : Coursera.org. Minimisation du coût, 1. (2014). [image mise à disposition par l'université de Stanford à des fins pédagogiques sur Coursera.org (Introduction to Machine Learning), 2015]

La valeur initiale de la matrice Θ est distinguée en mauve pour les deux entrainements réalisés sur la fonction de coût $J(\Theta)$.

L'utilisation d'une descente de gradient mini-batch pourrait encore aider à multiplier les chances de voir converger l'optimisation à proximité immédiate du minimum optimal. Cette méthode est, en effet, efficace par ses déplacements plus approximatifs ce qui lui permet parfois de sortir de minimum locaux lorsque ceux-ci sont peu profonds.

h) Structure du réseau neuronal

La structure idéale d'un réseau neuronal est toujours en débat dans de nombreuses universités et centres de recherche.

De manière générale, on s'accorde sur le fait que le dimensionnement des couches d'entrée et de sortie. Le nombre d'unités d'entrées correspond au nombre de variables associées à nos données auxquelles on ajoute un facteur « bios ». Le nombre d'unités de sortie est défini par le nombre de classes, c'est-à-dire le nombre de résultats différents potentiellement retournés par l'algorithme.

A l'inverse, il semble qu'il n'y ait pas encore de consensus sur le nombre et la composition des couches cachées. Plus le nombre d'unités et de couches augmente, meilleurs sont les résultats. Cependant, plus le réseau neuronal est dense, plus celui-ci demande des ressources pour être calculé. Il est donc nécessaire de réaliser un compromis entre ces deux contraintes.

Les travaux du professeur Andrew Ng tendent à recommander une approche graduelle.

Commencer avec une couche unique cachée semble être un bon début.

Si les résultats sont insuffisants, ajouter d'autres couches cachées en veillant à ce qu'elles comportent le même nombre d'unités que la première avant d'éventuellement modifier leur nombre.

A titre informatif, l'un des premiers réseaux neuronaux à gagner de la notoriété est le réseau NetTalk[SEJNOWSKI 1987] qui effectue la lecture (à haute voix) d'un texte en anglais. Ce réseau a appris à lever les ambiguïtés de syllabes s'écrivant de la même manière mais se prononçant différemment.

La couche d'entrée se compose de 7 caractères x 29 cellules, soit 203 cellules, chaque caractère étant non compressé et codé sur 29 bits).

La couche cachée est unique et composée de 80 unités.

La couche de sortie comporte, quant à elle, 26 cellules. Ces unités retournent les caractéristiques des phonèmes à émettre comme la zone vibratoire (labiale, dentale, etc.), le type de phonème (arrêt, nasale, etc.), la hauteur des voyelles, la ponctuation, etc.

Au total, ce réseau comporte 309 unités et 18320 connexions. NetTalk retourne 95 % des mots appris correctement et 75 % des nouveaux mots.

Citons à ce propos Jean-Pierre Nadal, Directeur de Recherche au CNRS & Directeur d'Etudes à l'EHESS de Paris : « Dans ses conférences, T. Sejnowski[14] faisait entendre à l'auditoire un

[SEJNOWSKI 1987] Sejnowski Terrence J. et Rosenberg Charles R., *NETtalk: a parallel network that learns to read aloud*, http://cs.union.edu/~rieffelj/classes/2009-10/csc320/readings/Sejnowski-speech-1987.pdf, 1987, consulté le 17 mars 2016

[14] Terry Sejnowski est un pionnier dans les domaines de « Computational Neuroscience » et « Machine learning ». Il est aujourd'hui professeur à l'université de Californie, San Diego

enregistrement sonore pris à divers moments au cours de la phase d'apprentissage. On pouvait alors entendre le réseau d'abords balbutier, puis on distinguait un découpage du texte en phrases, jusqu'à finalement une lecture raisonnable du texte. L'effet est évidemment spectaculaire, et il n'y a pas de doute, qu'à la suite de ces démonstrations, nombreux sont ceux qui se sont convertis au connexionnisme, si je puis dire... On a ainsi vu, et ceci principalement aux Etats-Unis, se développer la vague, née en 1985, d'une activité impressionnante de ceux pour qui, « ça y était »: pour résoudre n'importe quel problème, il suffit de mettre dans une boîte noire quelques neurones artificiels, d'injecter une base de données et de laisser tourner la « backprop » pendant une nuit ; au matin, miracle, on retrouve une machine intelligente. Comme l'a dit Yann Le Cun[15], l'un des inventeurs de l'algorithme, l'usage de la rétropropagation du gradient est à la fois « wide and wilde » (large et sauvage)... »[DENIS]

5. SVM, machine à vecteurs de support (Support Vector Machine)

Comme les réseaux neuronaux, les machines à vecteurs de support (parfois nommés séparateurs à vaste marge) sont un ensemble de méthodes pouvant résoudre des problèmes de régression ou de classification. Celles-ci ont été développées par Vladimir Vapnik, un mathématicien et professeur russe, ayant enseignés ses théories dans les universités de Moscou, de Londres et de Columbia (New York). Comme beaucoup des grands noms du « machine learning », il prête ses services à Facebook depuis 2014.

a) La fonction de coût

Les SVM reposent, d'abord, sur l'idée de simplifier la fonction de coût permettant l'optimisation d'une hypothèse en remplaçant les logarithmes par des fonctions conditionnelles plus simples[CHANG 2011].

Pour rappel, la formule de coût utilisée en régression logistique est la suivante :

[15] Yann Le Cun est un chercheur français s'étant particulièrement distingué pour ses contributions au deep learning. Il a rejoint le centre de Facebook en 2013.

[DENIS] Denis F. et Gilleron R., *Apprentissage à partir d'exemples*, http://www.grappa.univ-lille3.fr/polys/apprentissage/sortie005.html, date non trouvée, consulté le 16 mars 2016

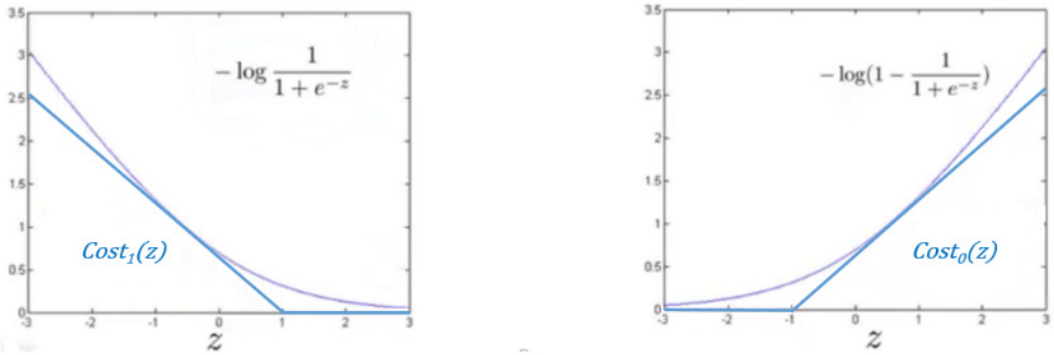
[CHANG 2011] Chang Chih-Chung et Lin Chih-Jen, *LIBSVM : a library for support vector machines. ACM Transactions on Intelligent Systems and Technology*, 2:27:1--27:27, 2011
http://www.csie.ntu.edu.tw/~cjlin/libsvm

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log \frac{1}{1+e^{-\theta^T x^{(i)}}} + \left(1-y^{(i)}\right) \log\left(1 - \frac{1}{1+e^{-\theta^T x^{(i)}}}\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

$$= \frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \left(-\log \frac{1}{1+e^{-\theta^T x^{(i)}}}\right) + \left(1-y^{(i)}\right)\left(-\log\left(1 - \frac{1}{1+e^{-\theta^T x^{(i)}}}\right)\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

La simplification appliquée sur cette fonction tient compte de l'état binaire de $y^{(i)}$ et part du principe que l'un des deux termes s'annule suivant la valeur $y^{(i)}$ (étant fixée à 0 ou 1, dans nos données d'entrées) :

$\left(1-y^{(i)}\right) \log\left(1 - \frac{1}{1+e^{-\theta^T x^{(i)}}}\right) = 0$, si $y^{(i)} = 1$ $\qquad$ $y^{(i)} \log \frac{1}{1+e^{-\theta^T x^{(i)}}} = 0$, si $y^{(i)} = 0$



Source : Coursera.org. Simplification, 1. (2014). [image mis à disposition par l'université de Stanford à des fins pédagogiques sur Coursera.org (Introduction to Machine Learning), 2015, annotée ensuite par Powerpoint]

La formule de coût obtenue après cette première simplification prend la forme suivante :

$$J(\theta) = \frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \operatorname{cost}_1(\theta^T x^{(i)}) + \left(1-y^{(i)}\right) cost_0(\theta^T x^{(i)})\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

Les multiplicateurs m de la fonction de coût réservée aux régressions logistiques sont également supprimés, ceux-ci n'intervenant en rien dans le calcul du vecteur optimal θ.

$$J(\theta) = \frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \operatorname{cost}_1(\theta^T x^{(i)}) + \left(1-y^{(i)}\right) cost_0(\theta^T x^{(i)})\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

Le paramètre λ est, enfin, remplacé au profit de C sur le premier terme. Celui-ci remplit le même rôle à savoir de prioriser un terme au détriment de l'autre plus de variance ou plus de régularisation (=> plus de biais)

$$J(\theta) = C[\sum_{i=1}^{m} y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T x^{(i)})] + \frac{1}{2}\sum_{j=1}^{n} \theta_j^2$$

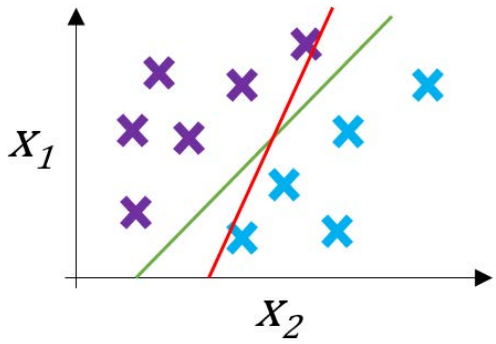
où $C = \frac{1}{\lambda}$, plus celui-ci augmente, plus on s'autorise de la variance en dessinant la frontière de décision.

Une fois optimisée, la fonction hypothèse retournera 1 si $\theta^T x \geq 1$, et 0 si $\theta^T x < -1$ avec la contrainte que $\theta^T x$ ne retourne jamais une valeur entre -1 et 1.

b) Large marge et régularisation

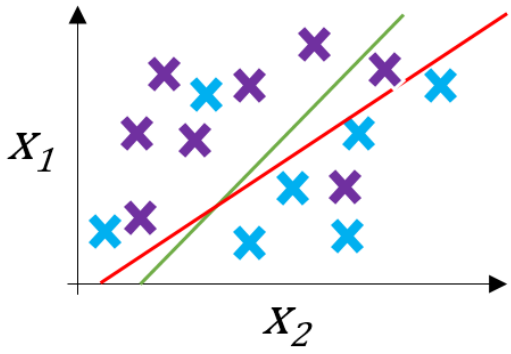
Le SVM est un séparateur à large marge. Cela signifie qu'il fixe la frontière de séparation entre les cas positifs et négatifs, autant que possible, à équidistance de ces données. Les données utiles au tracé de la frontière de décision sont appelés les **vecteurs de support**.

Sur l'illustration ci-dessous, on remarque que les frontières de décisions rouge et verte ne donnent lieu à aucune erreur de classification. On distingue, cependant, que la frontière verte semble épouser plus proprement la frontière naturelle qui se dessine entre les ensembles de croix bleues et mauves.



Représentation schématique de la fixation d'une frontière de décision, 1. [image inspirée d'une esquisse du cours en ligne de Coursera.org (Introduction to Machine Learning), 2015]

Ce comportement naturel de SVM est dû au paramètre de régularisation dont l'influence peut être modulée par ajustement sur le facteur C. Augmenter C favorise la variance. Une valeur élevée de C pourrait mener au résultat ci-dessous en rouge. Le diminuer tend à augmenter la marge.



Représentation schématique de la fixation d'une frontière de décision, 2. [image inspirée d'une esquisse du cours en ligne de Coursera.org (Introduction to Machine Learning), 2015]

c) Explication mathématique du calcul vectoriel du terme de régularisation

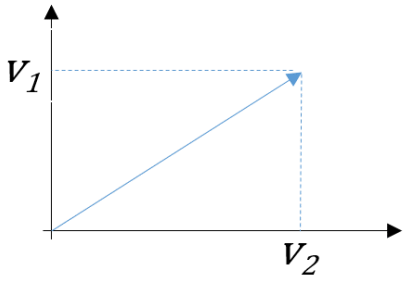
Comme mentionné précédemment, la fonction d'hypothèse inclus le paramètre de régularisation $\frac{1}{2}\sum_{j=1}^{n}\theta_j^2$. Lors de l'optimisation de cette fonction, ce paramètre tend à être minimisé par l'ajustement de θ.

Le calcul vectoriel s'avère très efficace pour ce calcul où la somme des variables du vecteur θ est remplacée par la norme au carré de celui-ci.

$$\frac{1}{2}\sum_{j=1}^{n}\theta_j^2 = \frac{1}{2}({\theta_0}^2 + {\theta_1}^2 + \cdots + {\theta_n}^2) = \frac{1}{2}\left(\sqrt{{\theta_0}^2 + {\theta_1}^2 + \cdots + {\theta_n}^2}\right)^2 = \frac{1}{2}\|\theta\|^2$$

Cette transformation s'explique en calcul vectoriel par le fait que la norme permette de calculer la longueur d'un vecteur au départ de ses coordonnées par simple application du théorème de Pythagore.

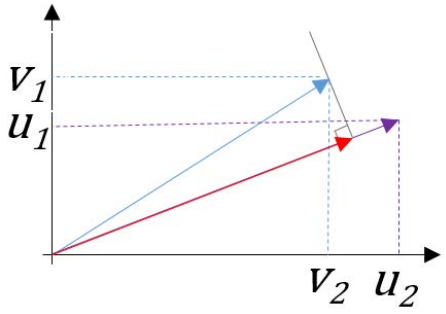
$$\|v\| = \sqrt{v_1^2 + v_2^2}$$



Norme en calcul vectoriel, 1. (2016).

Par ailleurs, le produit vectoriel de deux vecteurs équivaut à la projection à 90° d'un vecteur sur le second multipliée à la norme de ce dernier.

$$u^T v = p.\|u\|$$



Projection en calcul vectoriel, 1.

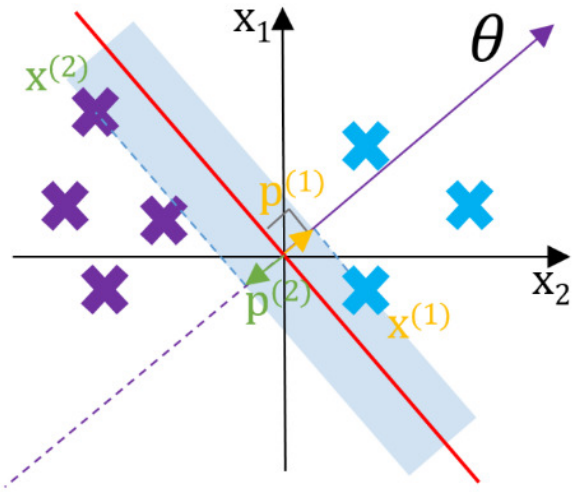
Ce produit vectoriel est utilisé par SVM pour minimiser $J(\theta)$, d'une part, sur le paramètre de régularisation et, d'autre part, sur l'évaluation de l'hypothèse $\theta^T x$ (où le vecteur x est projeté sur le vecteur θ) comme c'est le cas avec d'autres méthodes de discrimination ou de régression.

$$\theta^T x^{(i)} = p^{(i)}.\|\theta\|$$

où $p^{(i)}$ est la projection de x sur θ.

L'intérêt de son utilisation réside dans sa faculté à maximiser la marge entre deux ensembles en dessinant la frontière la plus éloignée possible des données d'entrée. Cela est un effet de bord positif de la minimisation de θ.

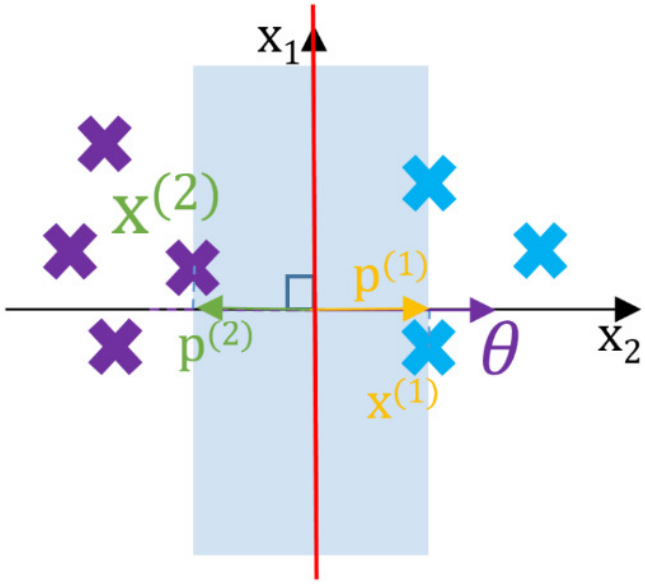
Prenons un exemple graphique où, pour simplifier, θ contiendrait 3 valeurs et où $\theta_0 = 0$. La frontière de décision (ici en rouge) a été déterminée de manière peu optimale entre les ensembles de données positives et négatives représentés par les croix bleues et mauves. Sachant qu'il est possible de démontrer mathématiquement que le vecteur θ est perpendiculaire à la frontière de décision. Celui-ci est représenté en mauve sur la figure ci-dessous. $p^{(i)}$ étant la projection de x^i sur θ, j'ai représenté les vecteurs $p^{(1)}$et $p^{(2)}$ étant respectivement les projections de $x^{(1)}$ et $x^{(2)}$ sur θ.



Faible marge, 1. [image inspirée d'une esquisse du cours en ligne de Coursera.org (Introduction to Machine Learning), 2015]

Comme il existe une contrainte stipulant que $y^{(i)} = 1$ si $\theta^T x \geq 1$, et $y^{(i)} = 0$ si $\theta^T x < -1$, on peut affirmer que $p^{(i)}.\|\theta\| \geq 1$ si $y^{(i)} = 1$, et $p^{(i)}.\|\theta\| < -1$ si $y^{(i)} = 0$.

Pour respecter cette contrainte, il faut que $p^{(i)}$ soit grand pour que $\|\theta\|$ soit petit et ainsi respecter l'effort de minimisation. On remarque alors que la marge marquée en bleu clair est bien plus large que ci-dessus.



Marge large, 1. [image inspirée d'une esquisse du cours en ligne de Coursera.org (Introduction to Machine Learning), 2015]

d) Noyau

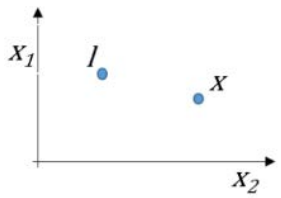
Dans la section relative aux régressions logistiques, nous avons vu qu'il était possible de représenter mathématiquement un ensemble de données. Cette représentation était obtenue au moyen d'une équation polynomiale dont les multiplicateurs ont été ajustés adéquatement.

Lorsque qu'il est question de déterminer la similarité de données ne partageant pas forcément de caractéristiques communes, la question se complexifie encore davantage. SVM tente d'apporter une solution à ce problème par l'utilisation d'une fonction spécifique appelée fonction de noyau (*kernel*)[MATH_TOULOUSE 2014]. Cette fonction peut, par exemple, prendre la forme d'une fonction linéaire ou polynomiale pour augmenter le nombre de dimensions comme abordé précédemment. Elle peut encore se définir par une fonction de similarité de noyau gaussien.

La fonction de similarité de noyau gaussien s'exprime comme suit :

$$similarity(x, l) = \exp(-\frac{\|x - l\|^2}{2\sigma^2})$$

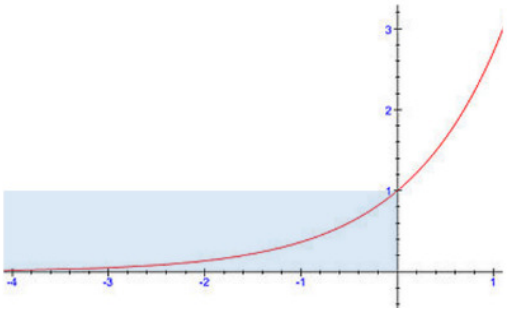
où x est la donnée dont on veut mesurer la similitude au point de référence l, la fonction exp() est la fonction exponentielle.



Deux données sans caractéristique commune, 1.

Si la similitude est très forte entre x et l, cette fonction retourne une valeur proche de 1 car $x - l$ est proche de 0. Si les variables de x et l sont, à l'inverse, éloignées, la fonction retourne une valeur proche de 0.

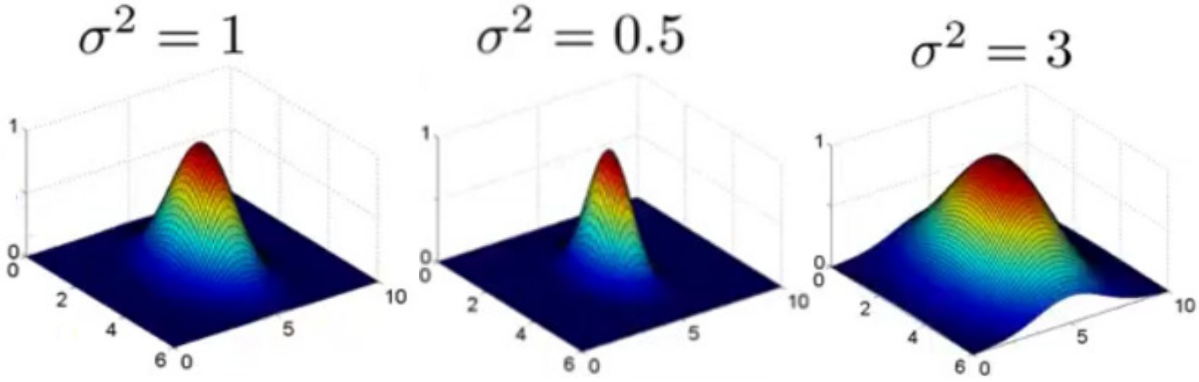
[MATH_TOULOUSE 2014] Département de Mathématiques de l'université de Toulouse, *Machine à vecteurs de supports*, http://www.math.univ-toulouse.fr/~besse/Wikistat/pdf/st-m-app-svm.pdf, p. 1-16, 19 avril 2014, consulté le 12 avril 2016



Fonction de similarité, 1.

Seules des valeurs inférieures ou égales à zéro peuvent être retournée par cette fonction (cf. zone bleue clair).

Le paramètre σ (Sigma) intervient sur la fonction de similarité en amplifiant sa courbure (si σ^2 augmente) ou en la réduisant.



Source : Coursera.org. Exemple d'adaptation de Sigma, 1. (2014). [image mis à disposition par l'université de Stanford à des fins pédagogiques sur Coursera.org (Introduction to Machine Learning), 2015]

e) Entrainement

La fonction de similitude n'est initialement pas issue du calcul vectoriel mais peut-être appliquée pour chaque variable d'une donnée si cela est souhaité. Celle-ci retourne alors un vecteur, noté $f^{(i)}$.

Avec les machines à vecteurs de support, l'optimisation de la fonction d'hypothèse se réalise comme suit :

$$h_\theta(x) = C\sum_{i=1}^{m} y^{(i)} cost_1\left(\theta^T f^{(i)}\right) + (1 - y^{(i)}) cost_0(\theta^T f^{(i)}) + \frac{1}{2}\sum_{j=1}^{n} {\theta_j}^2$$

où $n = m$, ce qui donne lieu à différentes optimisations du paramètres de régularisation (parfois $\theta^T\theta$ est remplacé par $\theta^T M\theta$ ou le second vecteur est multiplié par une constante pour des questions d'efficacité).

Pratiquement, les bibliothèques implémentant les SVM masquent ces aspects mathématiques. Les seules paramètres accessibles étant le facteur d'apprentissage C (qui remplace l'effet de λ) qui modifie la variance et la fonction de noyau accompagnée de son paramétrage. Dans le cas d'un noyau Gaussien, par exemple, il faut accompagner la fonction du paramètre σ qui détermine le seuil de similitude entre les données. Nous pourrions également faire le choix de n'utiliser aucune fonction de noyau. Dans ce cas, on parle de noyau linéaire.

Il est encore possible d'utiliser d'autres types de fonction de noyau comme des formes polynomiales, accompagnée ou non d'une constante ajoutée aux termes ($k(x, l) = (x^T l + constante)^{degré}$) ou d'autres formes standardisées pour des besoins précis comme le « String kernel » qui opère sur des séquences finies de caractères n'ayant pas la même longueur.

Conformément aux bonnes pratiques vues précédemment, il est important de fixer son choix sur le type de fonction de noyau qui donne les meilleurs résultats sur l'ensemble de données de validation (cross-validation test set).

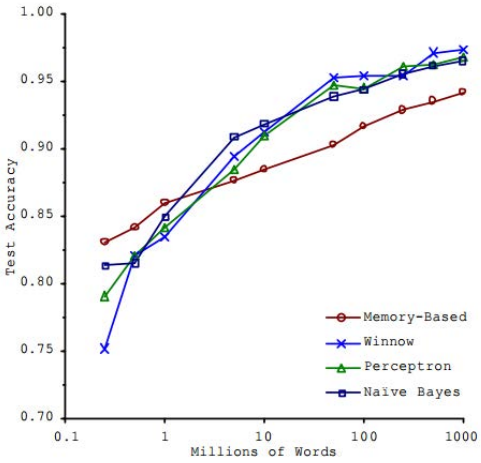
6. Quelle méthode choisir ?

Plusieurs méthodes d'apprentissage automatique viennent d'être présentées. Comme celles-ci poursuivent des objectifs communs, il est légitime de s'interroger sur laquelle utiliser.

Un travail de recherche[BANKO 2001] réalisée en 2001 par les ingénieurs Banko et Brill travaillant pour Microsoft n'a pas permis de répondre parfaitement à cette question dans le cadre d'une étude tentant de sélectionner le meilleur algorithme pour « désambiguer » le langage. Leur

[BANKO 2001] Michele Banko et Eric Brill, *Scaling to Very Very Large Corpora for Natural Language Disambiguation,* http://ucrel.lancs.ac.uk/acl/P/P01/P01-1005.pdf, 2001, consulté le 16 février 2016

conclusion était que dans l'essentiel des cas, le volume des données d'entrée importait bien plus que l'algorithme choisi.



Source : Michele Banko et Eric Brill, *Scaling to Very Very Large Corpora for Natural Language Disambiguation. (2001).* http://ucrel.lancs.ac.uk/acl/P/P01/P01-1005.pdf

Dans le cadre d'une classification, Andrew Ng, professeur de « Machine learning » à l'université de Stanford, recommande, quant à lui, de sélectionner l'algorithme en fonction du nombre de données en notre possession, noté m, et du nombre de leurs variables, noté n.

Si n est important au regard de m (par exemple $n = 10000$ et $m = 1000$), il suggère de tenter une régression logistique ou un SVM avec un kernel linéaire.

Si n est plutôt petit, inférieur à 1000, et que m n'atteint pas de valeur trop importante, il est alors possible d'utiliser pleinement SVM avec une fonction de kernel.

Si m atteignait une valeur importante, il faudrait revenir à l'utilisation d'une régression logistique ou un SVM avec un kernel linéaire (l'emploi d'une fonction de noyau demandant un temps de calcul important).

Les réseaux neuronaux se montrent compatibles avec toutes ces applications mais leur entrainement est plus coûteux. Cependant, bien que les résultats de l'étude des ingénieurs Banko et Brill étaient très proches entre ces méthodes en 2001, l'apprentissage profond (« deep

learning »), basé particulièrement sur les réseaux en couche comme les réseaux neuronaux, semble aujourd'hui se démarquer[MARTIN 2016] !

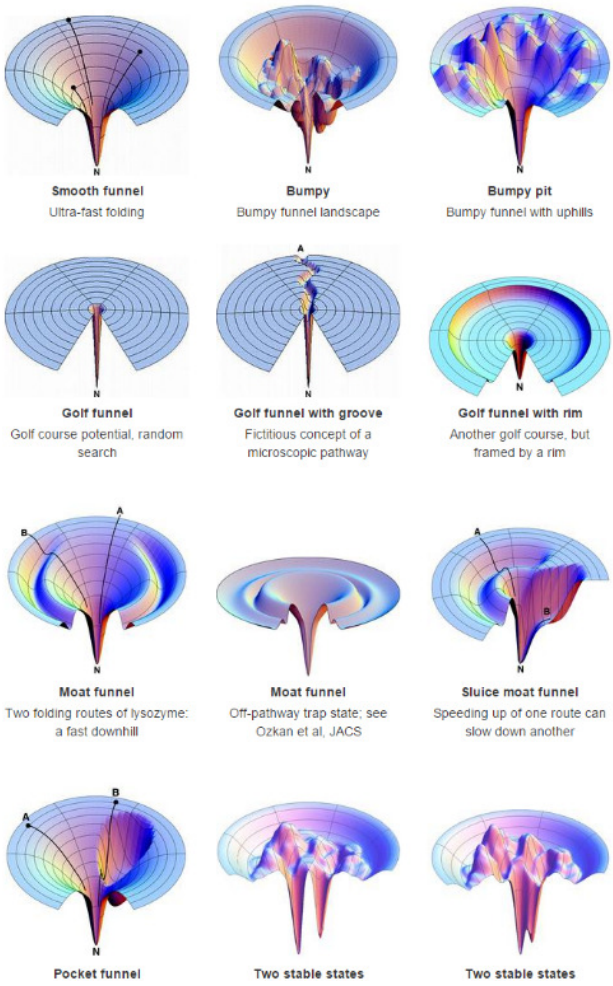
Celui-ci était pourtant tombé en désuétude dans les années 90, alors qu'on pensait que les réseaux neuronaux étaient pourvus de nombreux minima locaux ce qui rendait leur minimisation difficile et aléatoire. Les chercheurs associaient ce comportement à un autre phénomène issu de la physique, les verres de spin, une distribution d'impuretés magnétiques dans un alliage métallique. Sans entrer dans les détails, les verres de spin sont désignés comme frustrés car leur structure contient un grand nombre d'états métastables qui évoluent avec le vieillissement. Selon des chemins différents empruntés à la surface, deux spins peuvent donner des instructions contradictoires.

Les dernières recherches[CHOROMANSKA 2015] menées par Yann LeCun, directeur du laboratoire d'intelligence artificielle de Facebook et membre de l'Université, ou celles du Ken A. Dill, professeur à l'université Stony Brook de New York, semblent démontrer le contraire. La fonction de coût a, la plupart du temps, la forme d'un entonnoir pourvu, il est vrai, de plusieurs minima locaux mais ceux-ci sont bien souvent des points de selle (zone plus ou moins plate) et non de vraies cuvettes. L'ajout de cellules à un réseau neuronal a également tendance à rapprocher ces minima locaux, les uns des autres. Des techniques, comme celle de la descente du gradient stochastique, permettent bien souvent d'y échapper. Voilà qui relance l'enthousiasme autour du « *deep learning* ».

Les expérimentations menées par l'équipe de Yann LeCun ont été réalisées pour modéliser l'évolution du coût au regard du nombre d'unités, allant de 25 à 500, composant une couche cachée unique. Celles-ci mettent en évidence que des précautions particulières sont à prendre lors de la constitution du réseau neuronal. Plus celui-ci est dense, plus celui-ci tend à avoir une variance excessive (*overfitting*) et à rendre la recherche du minimum plus ardue.

[MARTIN 2016] Martin Charles H., *Why does Deep Learning work?,* https://charlesmartin14.wordpress.com/2015/03/25/why-does-deep-learning-work/, 25 mars 2015, consulté le 17 mars 2016

[CHOROMANSKA 2015] Choromanska Anna, Hena Mikael, Mathieu Michael, Ben Arous Gérard, Le Cun Yann, *The Loss Surfaces of Multilayer Networks*, http://arxiv.org/pdf/1412.0233.pdf, 21 janvier 2015, consulté le 17 mars 2016



Source : K.A. Dill, Energy Landscapes, http://dillgroup.stonybrook.edu/#/landscapes, 27 juillet 2015, consulté le 19 mars 2016.

IV. Partie pratique : comparaison des algorithmes

Cette section a pour objectif d'illustrer les notions théoriques présentées au chapitre précédent au moyen d'un cas pratique difficile à résoudre sans l'utilisation des techniques du *Machine Learning*. La mise en œuvre de différentes méthodes fera l'objet d'une étude comparative qui permettra de discuter de manière critique de leur intérêt dans ce cas d'espèce.

Comme stipulé préalablement, les techniques de *machine learning* ont de multiples utilisations parmi lesquelles on retrouve la reconnaissance optique de texte (*OCR*). Cette faculté est également appelée *Computer Vision*.

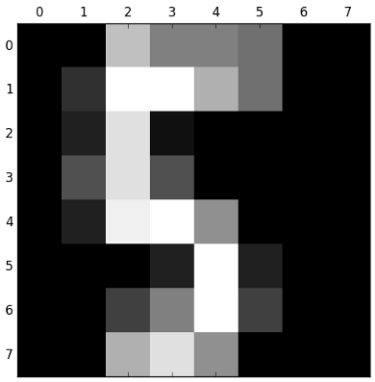
Dans l'exercice qui suit, les différentes méthodes de classification déjà abordées seront utilisées pour la construction d'un modèle de reconnaissance optique de caractères. Chacun de ces modèles pourrait ensuite être utilisé pour la reconnaissance de nouveaux caractères non rencontrés présentant les mêmes caractéristiques.

Les données sont issues d'un jeu de données mis à disposition par le projet Scikit-Learn[PEDREGOSA 2011] et intégré à leur distribution. Ce *dataset* se nomme digits[16] et contient 1797 caractères manuscrits, représentant les chiffres de 0 à 9, numérisés au format de 8 x 8 pixels et réduits en nuance de gris.

[PEDREGOSA 2011] Pedregosa & al., *Scikit-Learn: Machine Learning in Python*, JMLR 12, pp. 2825-2830, 2011

[16] http://scikit-learn.org/stable/auto_examples/datasets/plot_digits_last_image.html

A titre informatif, en agençant une donnée d'entrée en huit lignes et huit colonnes, il est possible de retrouver le caractère initial numérisé comme sur l'exemple suivant :



Vingt-cinquième caractère du dataset digits.

Tel que recommandé à la section « II.D.2 Phases d'évaluation de l'algorithme d'apprentissage automatique », le jeu de données est fragmenté en trois *datasets* de 60%, 20% et 20% respectivement pour les jeux d'entrainement, de validation et de test.

Je débuterai par les techniques de régression logistique. Celles-ci seront éprouvées avec la fonction *LibLinear* de *Scikit-Learn* et ainsi qu'avec la bibliothèque *Theano*[THEANO 2010], un projet collaboratif parallèle, lui aussi orienté sur le langage python, mais qui s'oriente plus particulièrement sur le *deep learning* au moyen de réseaux neuronaux.

Un exemple de réseau neuronal sera également réalisé avec cette même librairie.

Une implémentation de machines à vecteurs de support avec LibSVM[CHANG 2011] suivra.

Je conclurai enfin par un tableau comparatif des différents résultats et par un examen critique de la performance des méthodes.

[THEANO 2010] Bastien F., Lamblin P., Pascanu R., Bergstra J., Goodfellow I., Bergeron A., Bouchard N., Warde-Farley D. et Bengio Y.. *"Theano: new features and speed improvements", NIPS 2012 deep learning workshop*, 2012 ; Bergstra J., Breuleux O., Bastien F., Lamblin P., Pascanu R., Desjardins G., Turian J., Warde-Farley D. et Bengio Y.. *"Theano: A CPU and GPU Math Expression Compiler", Proceedings of the Python for Scientific Computing Conference (SciPy) 2010,* Austin TX, 30 juin au 3 juillet 2010

[CHANG 2011] Chang Chih-Chung et Lin Chih-Jen, *LIBSVM : a library for support vector machines. ACM Transactions on Intelligent Systems and Technology*, 2:27:1--27:27, 2011

A. Régression logistique avec Scikit-Learn

La régression logistique peut être réalisée au moyen de la fonction LogisticRegression[SKLEARN b], l'une des fonctions dédiées aux modèles linéaires intégrées à Scikit-Learn. Cette fonction offre de nombreux paramètres permettant d'ajuster la construction de son modèle comme le montre la partie théorique du mémoire à la section III.B.3 Régression logistique.

Dans ce premier exemple, j'ai testé plusieurs taux d'apprentissage avec les algorithmes de minimisation suivants : liblinear, newton-cg et lbfgs.

1. Entrainement

L'ensemble des poids initiaux de l'hypothèse de départ a été générée une première fois aléatoirement. Ces mêmes poids ont été ensuite utilisés pour tous les tests afin de préserver le cadre du modèle et permettre la comparaison non-biaisée des algorithmes.

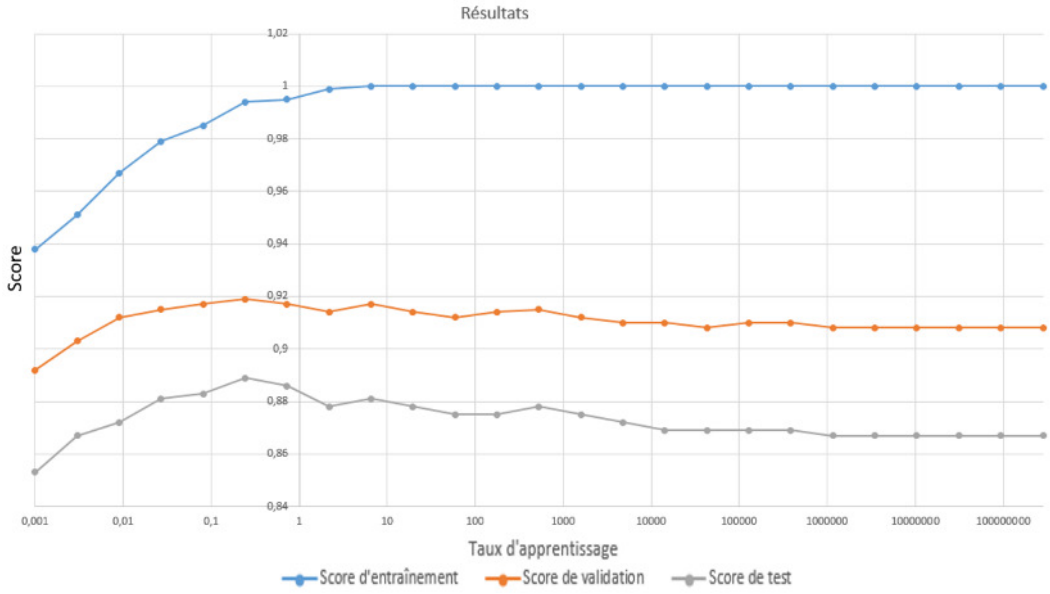
Dès le premier test avec LibLinear, on perçoit que le classificateur entrainé est capable de classifier correctement l'ensemble du jeu d'entrainement dès que le taux d'apprentissage atteint une valeur égale à 3. Ce phénomène est le résultat d'une limite à 1000 itérations paramétrée par mes soins dans l'objectif de rendre incontournable l'évaluation de différents taux d'apprentissage et de limiter par conséquent la durée de sa minimisation.

La courbe de résultat du jeu de validation indique qu'en l'état le classificateur généralise le mieux avec un taux d'apprentissage de 0,3.

Avec ce taux d'apprentissage, le jeu de données de test nous apprend que la précision de classification s'élève à 88,9%. Il reste donc 11,1% de classifications erronées.

Le graphique suivant illustre l'évolution du score (*accuracy*) en ordonnée avec les trois jeux de données au regard de l'ajustement du taux d'apprentissage en abscisse.

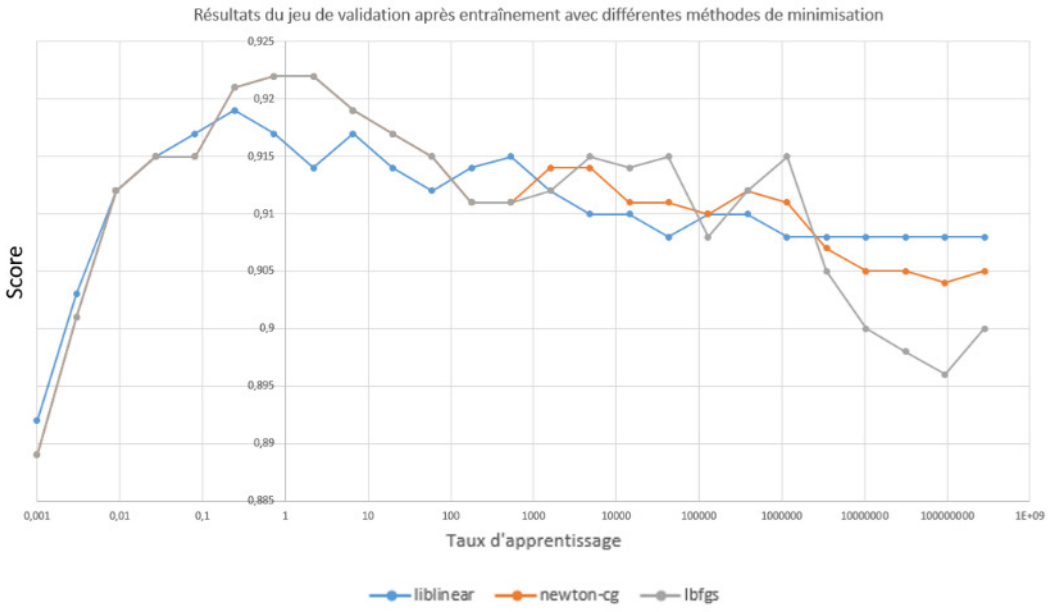
[SKLEARN a] Communauté de Scikit-Learn, *Documentation officielle de la version 0.17.1*, http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html, 2015, consulté le 4 avril 2016



Evolution du coût sur les trois jeux de données en fonction de l'ajustement du taux d'apprentissage.

Les algorithmes de minimisation différents offrent des performances légèrement différentes. On remarque que c'est l'algorithme LBFGS qui offre la meilleure performance sur le jeu de validation avec une précision de 92,2% et un taux de classifications correctes de 88,9% mesuré avec le jeu de test. Ce comportement est probablement attribuable au peu d'exemples présents dans le *dataset*. Sa composition engendre des courbes de résultat peu lisses. Avec davantage de données, les différentes méthodes de minimisation auraient conduit à des résultats plus proches encore.

Le graphique suivant montre l'évolution du score avec ces différentes méthodes de minimisation.



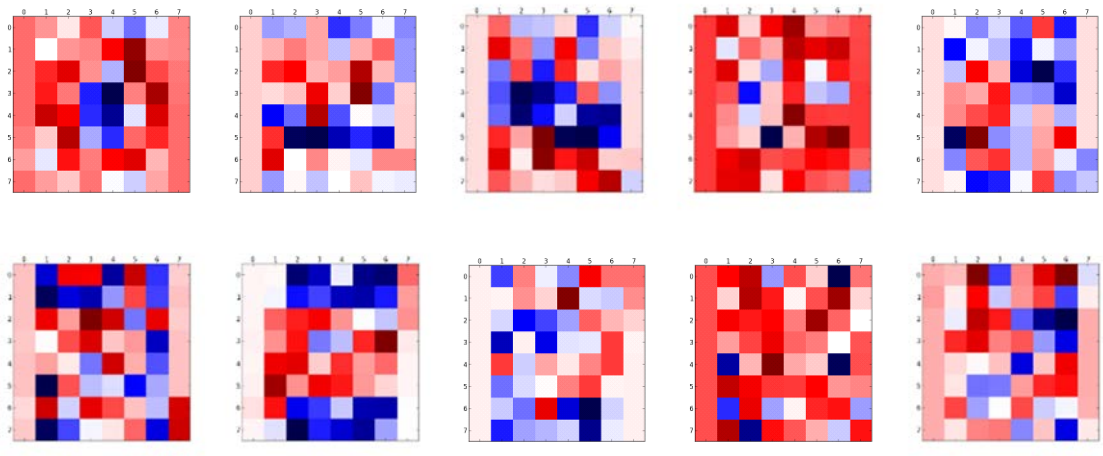
Evolution du coût avec les fonctions liblinear, newton-cg et lbfgs.

2. Visualisation des résultats

De l'entrainement, résultent dix fonctions mathématiques de 64 termes. Celles-ci sont composées des poids calculés pour chacun des pixels composant les images à classifier. L'ensemble des poids rend compte de l'importance de chaque pixel pour une classe donnée. Celui-ci est visualisable en exportant ces coefficients et en les réorganisant sur une matrice 8 x 8[SKLEARN b].

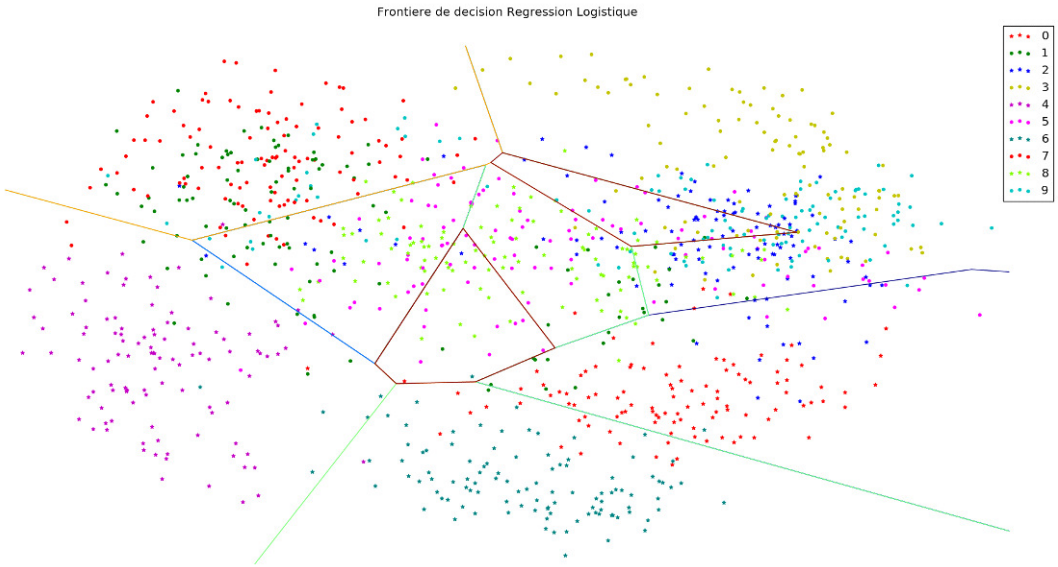
Sur les illustrations ci-dessous représentant les poids calculés pour les chiffres de 0 à 9, les pixels de coloration rouge sont ceux dont le poids est le plus important pour les distinguer des autres classes.

[SkLEARN b] Communauté de Scikit-Learn, *Documentation officielle de la version 0.17.1*, http://scikit-learn.org/stable/auto_examples/linear_model/plot_iris_logistic.html, 2015, consulté le 5 avril 2016



Poids des pixels à l'issue d'une Régression Logistique pour les chiffres de 0 à 9.

Aplati en deux dimensions au moyen de la méthode PCA, les frontières de décisions s'établissent comme le montre la figure suivante :



Frontières de décision à l'issue de la régression logistique linéaire représentées en deux dimensions au moyen de PCA.

Cette figure est dépourvue de notation d'abscisse et d'ordonnée, la réduction de dimensionnalité ayant projeté les données sur le plan en deux dimensions le plus adéquat. On perçoit que dix ensembles s'y dessinent très nettement et que les données d'une même classe, désignées par une couleur déterminée, se situent dans leur majorité à proximité immédiate les unes des autres. Cela signifie que, malgré le fait que la dimensionnalité soit réduite à deux dimensions, on perçoive encore des caractéristiques communes aux données d'une même classe.

Le fait que les 64 variables aient été transformées par projection comme expliqué à la section « III.B.2.c)(2) Utiliser moins de variables à la fois » sur un plan en deux dimensions n'a donc pas altéré totalement l'esprit de la classification.

Les frontières de chaque ensemble sont représentées sous forme de droite.

Toutes les données ne se situent cependant pas au sein du même ensemble alors que la phase d'entrainement affiche un score de classification (*accuracy*) de 100%. C'est une conséquence de la réduction. Cela démontre que, s'il avait été possible de représenter ces données en 64 dimensions, nous aurions perçu les ensembles comme étant plus éloignés qu'il n'y paraît sur cette représentation.

Il est possible d'étudier plus rigoureusement la classification au moyen des indicateurs de précision, de rappel et de score F-1 par classe avec le module « *sklearn.metrics* ».

Sur le jeu de données de test, on remarque par exemple que la précision atteinte par le modèle linéaire pour le chiffre 4 est de 100%. Cela signifie que toutes les données évaluées comme étant des 4 l'étaient réellement. En revanche, le rappel (*recall*) est de 92%, cela indique que certaines données représentatives du caractère 4 ont été catégorisées incorrectement.

	precision	recall	f1-score
0	0.92	0.94	0.93
1	0.76	0.81	0.78
2	0.97	0.97	0.97
3	0.92	0.65	0.76
4	1.00	0.92	0.96
5	0.83	0.95	0.89
6	0.95	0.97	0.96
7	0.94	0.89	0.91
8	0.76	0.88	0.82
9	0.79	0.84	0.82
avg / total	0.89	0.88	0.88

Score de précision, de rappel et F-1 pour le modèle de régression logistique linéaire

Plus précisément, ces résultats sont visibles au moyen d'une matrice de confusion. Tel que décrit à la section « II.D.3 Indicateurs de performance », cette matrice représente la classification effective des données de chaque classe et met en évidence les erreurs de classification.

Le tableau ci-dessous met, par exemple en évidence que le modèle classifie plutôt correctement les données de la classe 4 bien que trois occurrences aient été perçues que des 1 ou 9.

Par une lecture verticale de cette matrice, nous percevons en détail les résultats utiles au calcul de la précision et le rappel par une lecture horizontale.

```
Confusion matrix:
[[33  0  0  0  0  0  1  0  1  0]
 [ 0 29  0  1  0  0  0  0  1  5]
 [ 1  0 34  0  0  0  0  0  0  0]
 [ 0  1  0 24  0  5  0  2  5  0]
 [ 0  1  0  0 34  0  0  0  0  2]
 [ 0  1  0  0  0 35  1  0  0  0]
 [ 0  0  1  0  0  0 36  0  0  0]
 [ 0  1  0  0  0  0  0 32  2  1]
 [ 0  4  0  0  0  0  0  0 29  0]
 [ 2  1  0  1  0  2  0  0  0 31]]
```

Matrice de confusion (des classes de 0 à 9) pour le modèle de régression logistique linéaire

3. Introduction de la normalisation

L'introduction de la normalisation de variables dans une plage comprise entre 0 et 1 ou entre -1 et 1 n'amène aucune amélioration significative avec ce jeu de données. Cela est dû au fait que les données sont déjà dans un intervalle restreint. Les niveaux de gris sont, en effet, exprimés sur une échelle comprise entre 0 et 15.

4. Introduction de régularisation

En introduisant la régularisation, LibLinear obtient de meilleures performances affichant un score de 92,4% sur le jeu de validation et 89,7% sur celui de test. On peut donc conclure que le premier modèle entrainé comportait une variance excessive.

B. Régression logistique avec Theano

Comme stipulé plus haut, la bibliothèque Theano[THEANO 2016] a la particularité de permettre une implémentation aisée de réseaux neuronaux. L'étape actuelle, bien qu'elle semble être redondante avec la section précédente, sera donc utile a posteriori pour mettre en évidence

[THEANO 2016] Site Web du projet collaboratif Theano, *Documentation officielle*, http://deeplearning.net/software/theano/, 2016, consulté le 15 avril

l'intérêt de la transformation de la régression logistique en réseau neuronal par l'ajout d'une couche cachée.

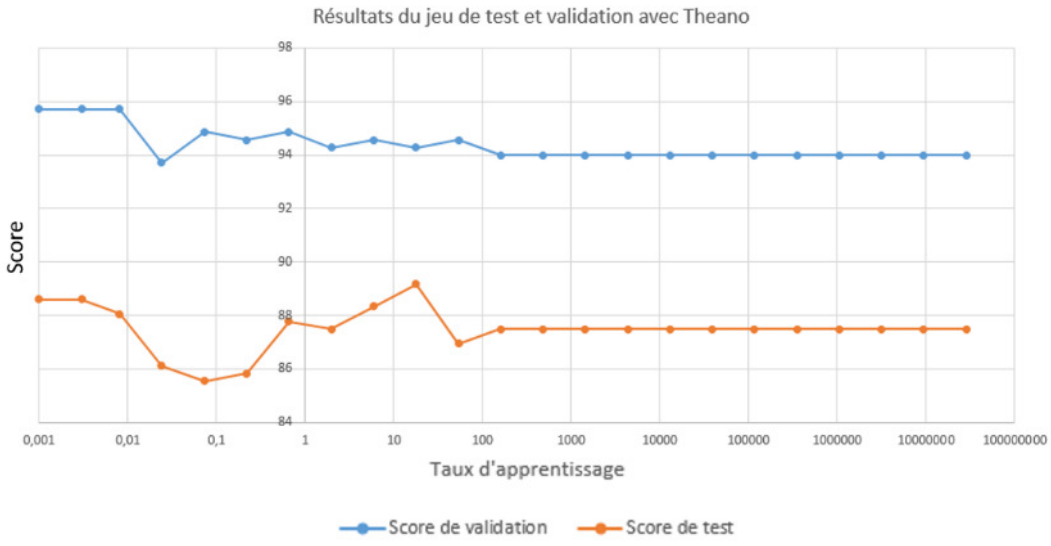
La bibliothèque Theano implémentée en Python compile en arrière-plan les fonctions déclarées en un code dans le langage C qui correspond à la représentation mathématique des fonctions qui y sont déclarées symboliquement. Cette génération a pour objectif de réaliser un programme exécutable autonome qui rend plus efficace l'utilisation après la phase d'entrainement, laquelle s'avère plus lente de par la compilation qu'elle intègre. Cette bibliothèque est également pourvue de méthodes permettant la construction et l'exécution du modèle mathématique sur GPU.

La réalisation d'un nouvel exercice au moyen de cette librairie me permet, au passage, de mettre en évidence l'utilité d'un gradient mini-batch. Comme indiqué à la section « II.C.3 Le gradient descendant mini-batch (*Mini-Batch Gradient Descent*) », le gradient mini-batch segmente le jeu de données initial en lots utilisés tour à tour pour entraîner l'algorithme. Celui-ci limite les ressources nécessaires (temps de calcul et consommation de mémoire) sans dégrader excessivement la qualité du modèle. Cette méthode permet, en outre, de sortir de minima locaux par sa trajectoire plus approximative.

Après ajustement du taux d'apprentissage, même avec une régression logistique classique, les résultats obtenus apparaissent légèrement plus prometteurs qu'avec Scikit-Learn. Le résultat affiche un score de 95,7% sur le jeu de validation et de 88,6% sur celui de test.

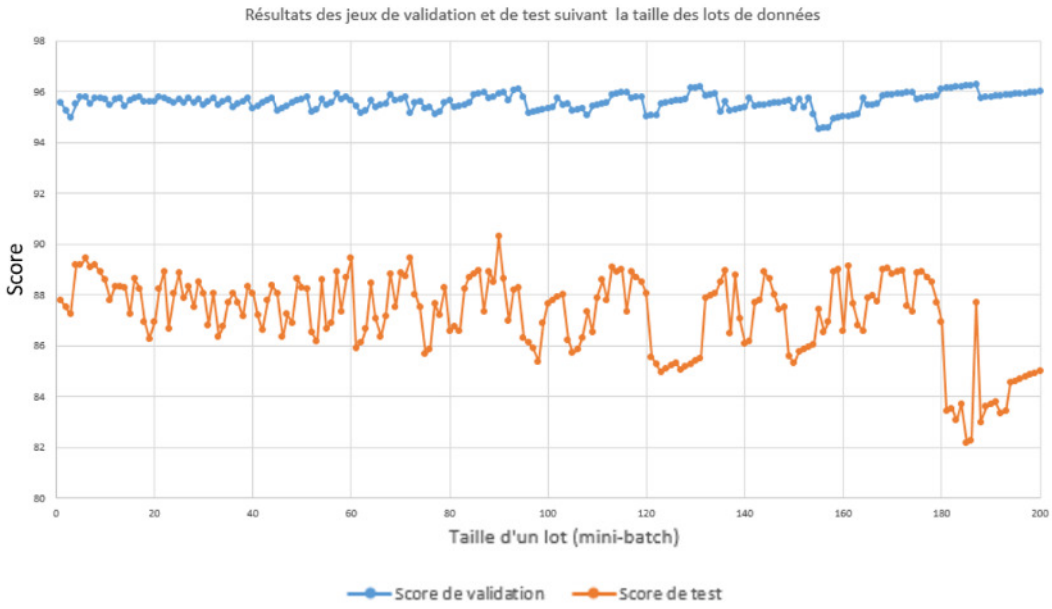
Cette amélioration pourrait s'expliquer par l'utilisation de la méthode de minimisation par gradient mini-batch. Il est fort probable que celle-ci ait permis de quitter un minimum local au profit d'un autre légèrement meilleur ce que n'ont pas permis les méthodes réalisées avec Scikit-Learn utilisant une descente de gradient classique. Comme dans l'exercice précédent, le taux de classification correcte est de 100% après ajustement du taux d'apprentissage avec le jeu d'entrainement.

Le graphique suivant affiche l'évolution du score obtenu en fonction du taux d'apprentissage utilisé.



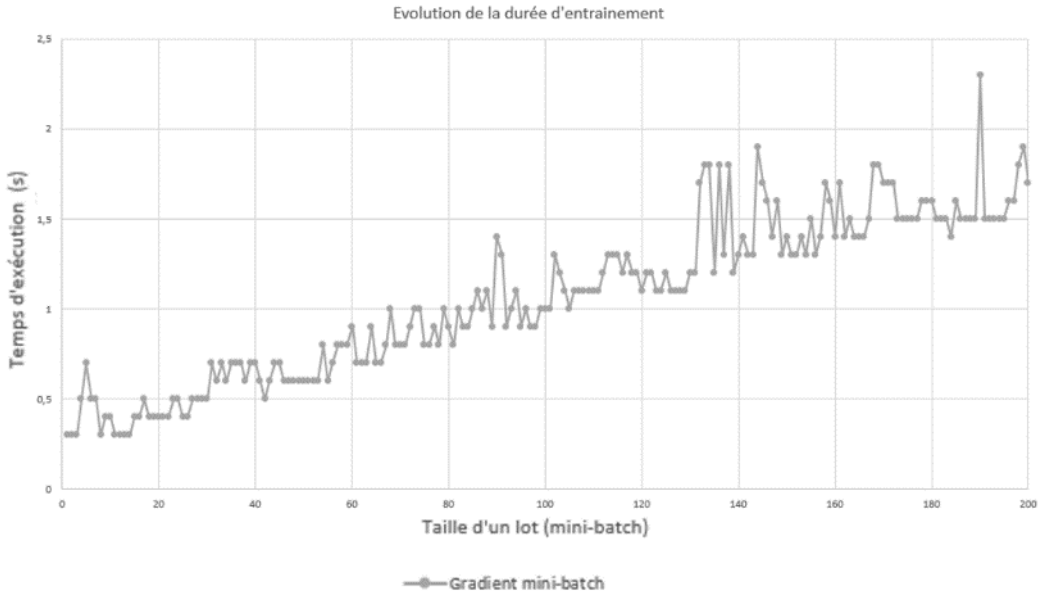
Evolution du coût avec la librairie Theano en minimisant l'hypothèse au moyen d'un gradient mini-batch de 10 éléments par lot.

Dans le cas de cet exercice, comme le montre le graphique suivant, le coût affiche globalement une tendance stable quel que soit la taille du lot utilisé. Cela s'explique par le fait que le *dataset* soit mélangé au préalable.



Evolution du score sur les jeux de données de validation et de test en fonction de la taille d'un lot de données avec mini-batch.

A l'inverse, comme présenté ci-dessous, le temps d'exécution montre une nette amélioration lorsque les lots manipulés sont de tailles restreintes.



Evolution du temps d'apprentissage sur le jeu de données d'entrainement en fonction de la taille d'un lot de données avec mini-batch.

Cette considération nous invite à procéder à un arbitrage entre précision et rapidité d'entraînement, selon le contexte d'utilisation, lorsque des contraintes existent à ce niveau.

C. Réseau neuronal avec Theano

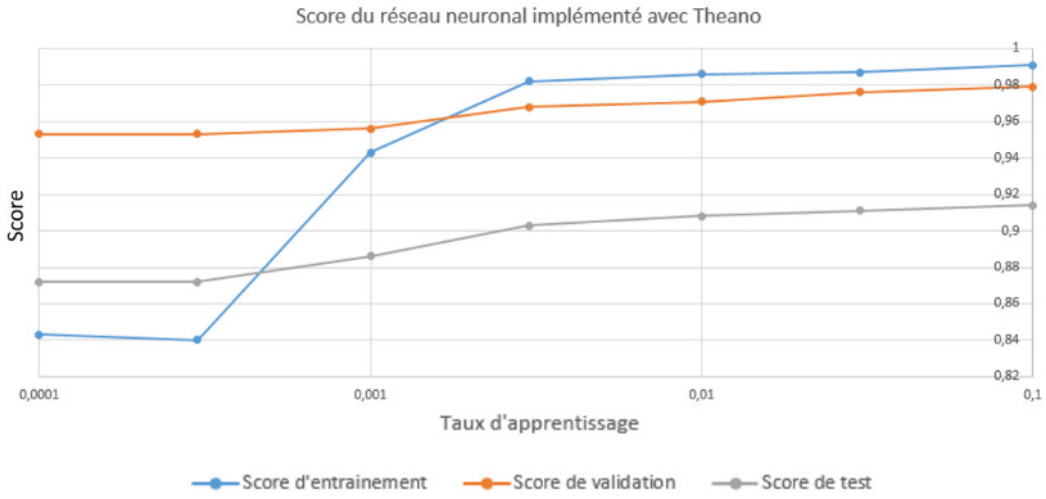
La troisième implémentation présentée dans ce mémoire est un réseau neuronal. La méthode suggérée dans le guide d'utilisation de leur outil est d'importer la régression logistique réalisée et d'y greffer une couche cachée entre la couche d'entrée et celle de sortie.

La structure d'un programme Python avec cette librairie s'écrit sous la forme d'un programme orienté objet. Le constructeur de la classe MLP (*Multi-Layer Perceptron*) possède un attribut *n_hidden* permettant de renseigner le nombre d'unités cachées sur cette couche unique. Schématiquement, chacune de ces unités est connectée à chaque unité de la couche d'entrée et à la couche de sortie. Les images possédant 64 pixels (*features*), la couche d'entrée comportera 64 unités. La classification menant à dix catégories (les nombres de 0 à 9), la couche de sortie compte 10 unités.

1. Entrainement

L'ensemble des poids initiaux est généré aléatoirement à l'aide d'une fonction *numpy.random.RandomState* issue de la librairie *numpy*. Cette étape garantit que les poids des différentes unités n'évoluent pas de manière identique.

Avec 500 unités cachées, le modèle prédit correctement 97,9% des données du jeu de validation et 91,4% du jeu de test avec un taux d'apprentissage fixé à 0,1, comme le montre le graphique suivant.

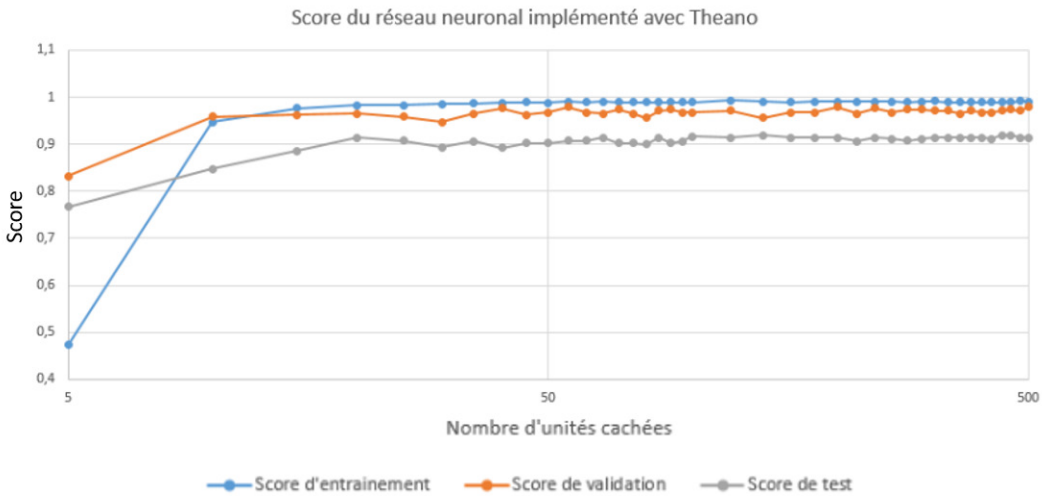


Evolution du score sur les jeux de données en fonction du taux d'apprentissage.

Comme évoqué à la section « III.B.4.h) Structure du réseau neuronal », plus le nombre d'unités composant la couche cachée est élevé, plus le réseau neuronal est précis. Dans l'exercice que je mène, on remarque cependant un net plafonnement. Les gains obtenus par l'ajout d'unités à un modèle qui en compte déjà une centaine deviennent insignifiants.

Cela s'explique par la relative pauvreté du jeu de données. Le nombre d'images ainsi que leur nombre de *features* est trop faible pour un réseau neuronal de 500 unités.

Le modèle semble généraliser relativement correctement. Les courbes des scores d'entrainement et de validation sont effectivement très proches comme le montre la figure suivante.



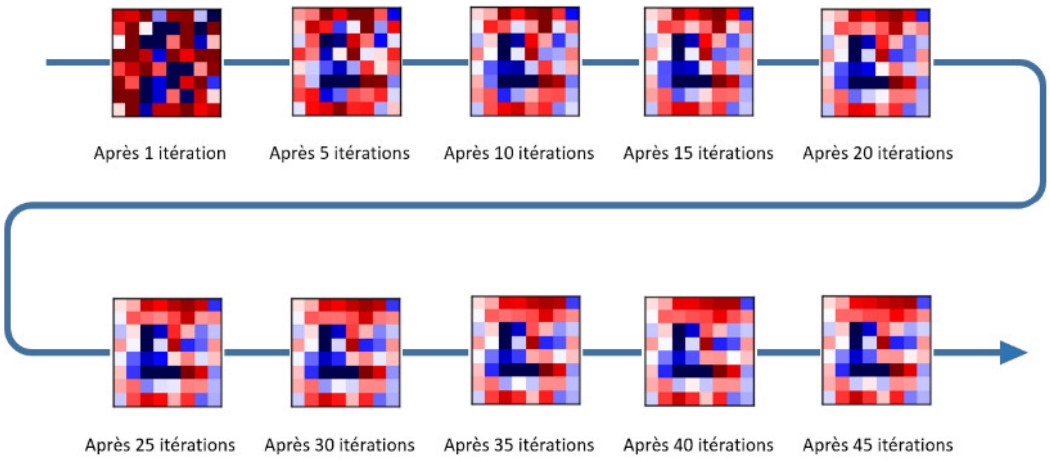
Evolution du score sur les jeux de données en fonction du nombre d'unités d'une couche cachée unique.

2. Visualisation de l'évolution des poids d'une unité cachée

L'entrainement d'un réseau neuronal n'est pas simple à visualiser. L'initialisation aléatoire des poids de la couche cachée engendre en effet une spécialisation différente de chaque unité dans la reconnaissance d'une portion ou de la totalité d'une image. L'utilisation des mêmes poids, définis une première fois aléatoirement, à plusieurs reprises permet néanmoins de relancer l'optimisation et de lire le poids de ces unités au fur et à mesure de l'entrainement lorsqu'on l'interrompt anticipativement[SKLEARN c]. De cette manière, il est possible de visualiser sous forme d'image les poids d'une unité au terme d'un nombre d'itérations données.

Sur l'image suivante, à titre d'exemple, après avoir sélectionné une unité cachée, il a été possible de mettre en évidence que celle-ci semble se spécialiser dans la reconnaissance du caractère « 3 ».

[SKLEARN c] Communauté de Scikit-Learn, *Documentation officielle de la version 0.17.1*, http://scikit-learn.org/dev/auto_examples/neural_networks/plot_mnist_filters.html, date inconnue, consulté le 16 avril 2016



Evolution des poids d'une unité de la couche cachée.

Pour rappel, la décision d'une unité cachée est une valeur légèrement supérieure à 0 ou légèrement inférieure à 1. Cette valeur est le résultat d'une fonction sigmoïde appliquée à un polynôme dont chaque terme est calculé par la multiplication du poids de la connexion et de la valeur de la variable de la donnée d'entrée.

Sur l'image ci-dessus, les pixels de couleur rouge sont ceux dont le poids est le plus important pour une unité de la couche cachée. Chacun de ces poids est multiplié au niveau de coloration du pixel correspondant de la donnée d'entrée. Plus le poids est élevé, plus celui-ci engendre un résultat élevé lors de la sommation du polynôme. Si plusieurs pixels sombres de la donnée d'entrée sont multipliés à des poids importants, il y a de fortes chances pour que le polynôme prenne une valeur positive. Inversement, la présence de pixels sombres de la donnée d'entrée multipliés à des poids négatifs (pixels colorés en bleu sur la matrice de poids), pénalise le résultat du polynôme et renforce la probabilité que le polynôme soit négatif.

La décision prise par l'unité se déterminant par la fonction sigmoïde de ce résultat, la présence dominante de pixels sombres dans les zones rouges et de pixels clairs dans les zones bleues provoque la décision en sortie d'une valeur proche de 1. A défaut, cette fonction retournera une valeur positive très proche de 0.

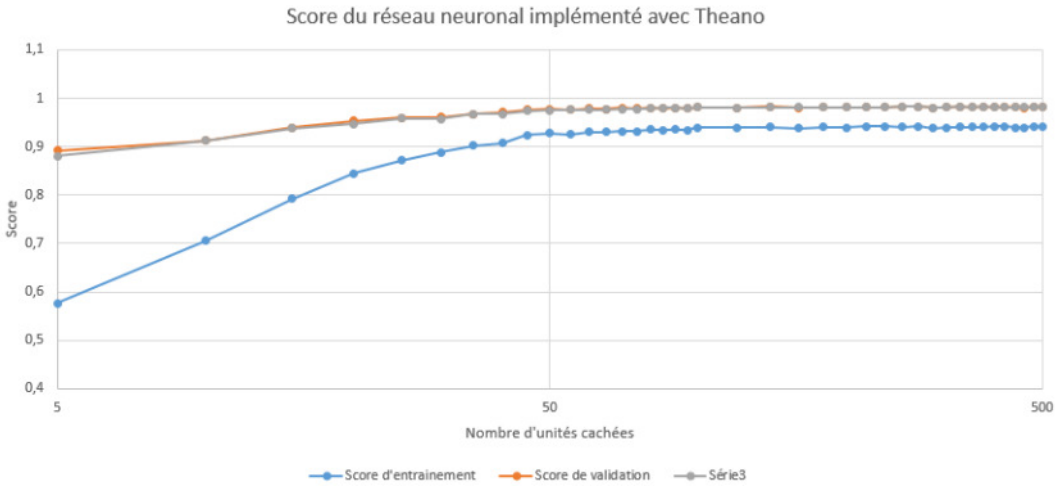
Comme cela est perceptible sur l'image suivante, les poids s'initialisent aléatoirement. L'entrainement modifie successivement les poids en entrée et en sortie de chaque unité de la couche cachée pour minimiser le nombre d'erreurs. Chaque unité s'adapte, bel et bien, pour se

spécialiser dans la reconnaissance d'une partie ou de la totalité d'une image telle que le montre l'illustration.

3. Comparaison avec le *dataset* MNIST

A titre comparatif, les résultats suivants ont été obtenus avec le même réseau neuronal sur base du *dataset* MNIST[17], un jeu de données de 70.000 caractères numérisés en niveaux de gris sous forme de matrice 28 x 28 pixels. Ce dataset contient donc approximativement 38 fois plus de caractères dotés de 12 fois plus de variables qu'en contient le dataset digits, soit un volume de données 477 fois supérieur. De manière analogue au dataset « digits » de Scikit-Learn, chaque image représente un caractère manuscrit compris entre 0 et 9.

Bien qu'il ne s'agisse pas rigoureusement des mêmes caractères numérisés (la comparaison est donc contestable), il est intéressant de remarquer que les résultats sont bien meilleurs et que l'ajout d'unités à la couche cachée offre de manière plus évidente un gain dans l'optimisation de l'hypothèse.



Evolution du score sur les jeux de données en fonction du nombre d'unités d'une couche cachée unique avec le dataset MNIST.

Après entrainement, ce réseau neuronal affiche un résultat de 98,3% de classification correcte sur les jeux de validation et de test.

[17] http://yann.lecun.com/exdb/mnist/

Aussi peut-on conclure que l'ajout d'unités à une couche cachée d'un réseau neuronal n'a d'intérêt qu'à la condition que la complexité du réseau aille de pair avec la complexité des données.

D. Réseau neuronal avec Scikit-Learn

En version de Scikit-Learn 0.18dev encore en développement intègre maintenant une classe permettant la réalisation de réseaux neuronaux[SKLEARN d] en décrivant le nombre de couches cachées et leurs constitutions.

Les nombreux essais réalisés avec cette classe n'ont cependant pas permis de mettre en évidence de gains même faibles en complexifiant le réseau neuronal. Au contraire, les résultats tendent à se dégrader.

Un réseau composé de deux conches cachées de 250 unités atteint, par exemple, un score de 100% avec le jeu d'entrainement, de 94,3% avec le jeu de validation et de 91,1% avec le jeu de test.

Un réseau composé de dix couches cachées de 50 unités atteint, quant à lui, un score de 100% avec le jeu d'entrainement, de 88,5% avec le jeu de validation et de 86,9% avec le jeu de test.

On peut conclure que le fait d'ajouter des couches cachées à un réseau neuronal renforce sa variance et que cette complexification n'ait de sens qu'avec un domaine plus complexe que la reconnaissance de caractères réduit en niveau de gris sur une matrice de 64 pixels.

Les réseaux convolutifs (*Convolutional Neural Networks*), parfois nommé LeNet ou ConvNet, pourraient néanmoins tirer parti de réseaux plus complexes[LENET 2016]. Ceux-ci sont principalement utilisés pour la reconnaissance d'images car ils spécialisent des groupes d'unités à la reconnaissance de motifs spécifiques dans des portions d'images géographiquement proches. Les unités des premières couches cachées sont volontairement connectées à un nombre de

[SKLEARN d] Communauté de Scikit-Learn, *Documentation officielle de la version 0.18.dev0*, http://scikit-learn.org/dev/modules/generated/sklearn.neural_network.MLPClassifier.html, 2016, consulté le 18 avril 2016

[LENET 2016] Site Web du projet collaboratif Theano, *Documentation officielle*, http://deeplearning.net/tutorial/lenet.html, 2016, consulté le 18 avril

variables limité représentant une zone restreinte de l'image. Les couches suivantes consolident et agrègent le réseau.

E. Machine à vecteurs de support avec LibSVM

1. Entrainement

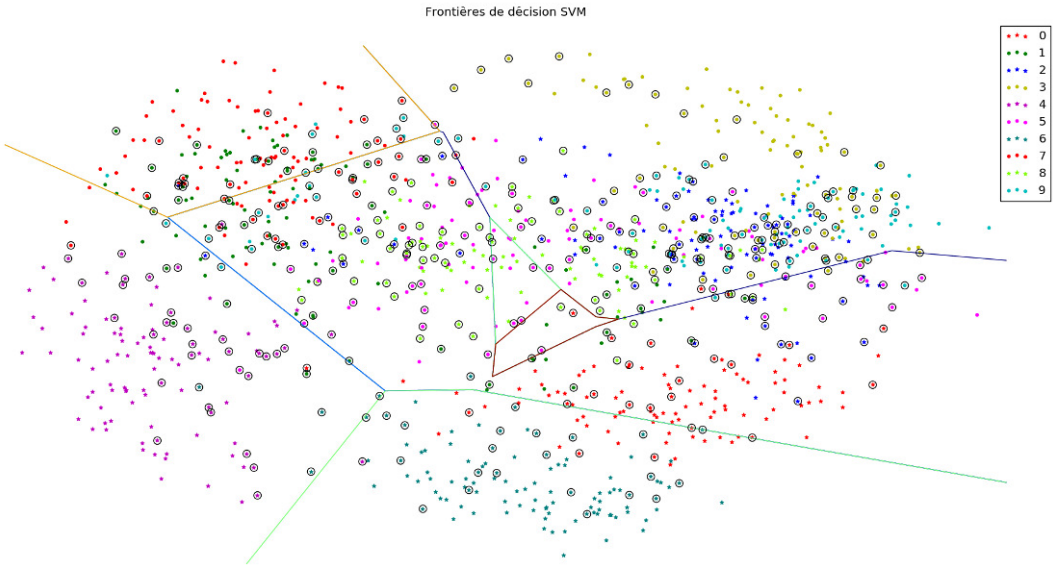
Entrainé avec le dataset, les machines à vecteurs de support démontrent également de très bonnes performances. Comme indiqué sur le graphique suivant, le meilleur résultat obtenu s'élève à 97,2% sur le jeu de validation et à 93,1% sur le jeu de test, faisant mieux qu'un réseau neuronal sur le jeu de test mais légèrement moins bien sur le jeu de validation.

SVM, Evolution du score sur les jeux de données en fonction du taux d'apprentissage.

2. Visualisation des résultats

Comme illustré en régression logistique, il est possible de visualiser les frontières de décision en réduisant la dimension d'affichage à deux axes[SKLEARN e]. Cette réduction est, à nouveau, réalisée par la fonction PCA qui aplatit le modèle sur le plan dont l'éloignement des données est le plus faible. Partant de la meilleure hypothèse et en diminuant la dimensionnalité de 64 à deux dimensions. L'aplatissement engendre cependant une perte sensible d'expressivité comme nous pouvons le constater sur la figure suivante.

[SKLEARN e] Communauté de Scikit-Learn, *Documentation officielle de la version 0.17.1*, http://scikit-learn.org/stable/auto_examples/svm/plot_iris.html, 2014, consulté le 20 avril

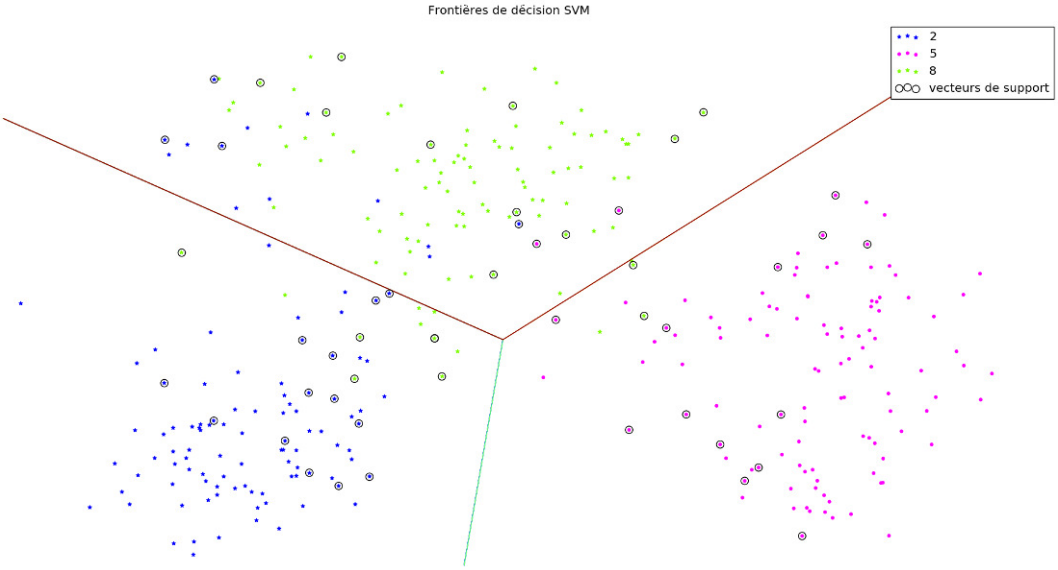


SVM, frontières de décision et vecteurs de support avec une machine à vecteurs de support linéaire.

Les vecteurs de support apparaissent ici entourés d'un cercle noir. Ces vecteurs incarnent les points de référence ayant servi à la fixation de la frontière de décision. Cette frontière est naturellement tracée à équidistance de ceux-ci.

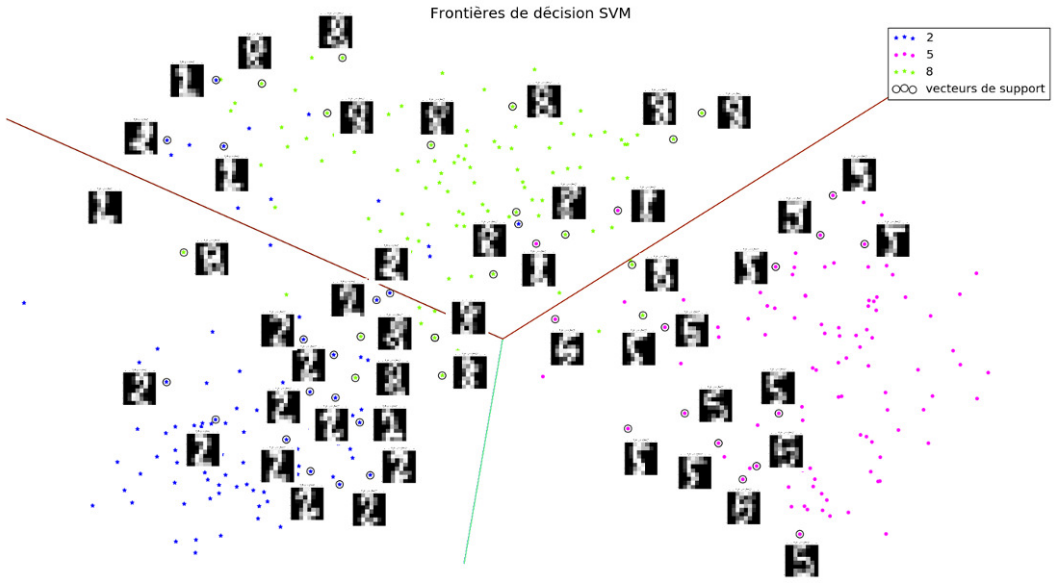
Les frontières ont l'apparence d'être formées de plusieurs segments rectilignes. C'est la conséquence d'une classification « un contre tous » avec un noyau linéaire.

Si nous réduisons les données affichées aux seuls chiffres 2, 5 et 8, on montre, sur l'image suivante, qu'aucune part de l'espace n'échappe à la classification et qu'aucune zone n'est donc non classifiée.



SVM, frontières de décision pour les chiffres 2, 5 et 8 et vecteurs de support avec un kernel linéaire.

En visualisant les images du jeu d'entrainement qui servent de vecteurs de support, on fait ressortir certaines similitudes. Il est cependant utile de faire remarquer qu'il est possible que des données, qui apparaissent proches sur cette image, soient éloignées avant l'aplatissement avec la fonction PCA.

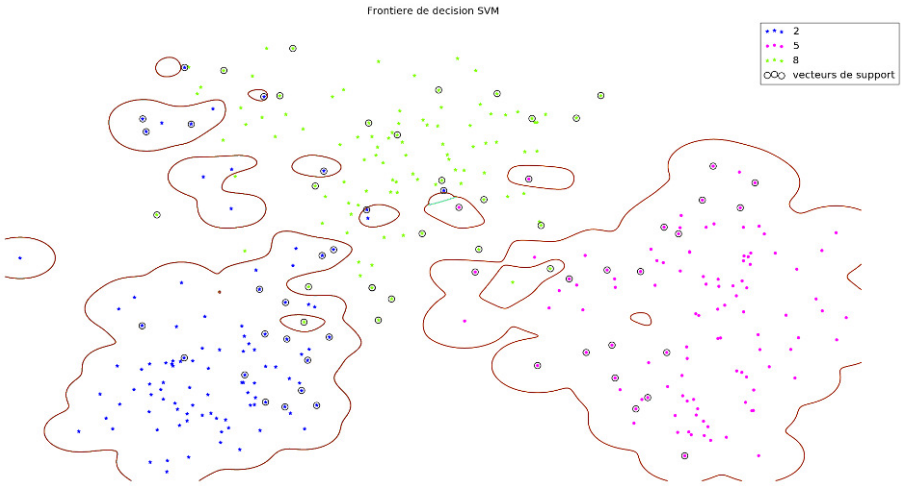


SVM, frontières de décision pour les chiffres 2, 5 et 8 et vecteurs de support, 2.

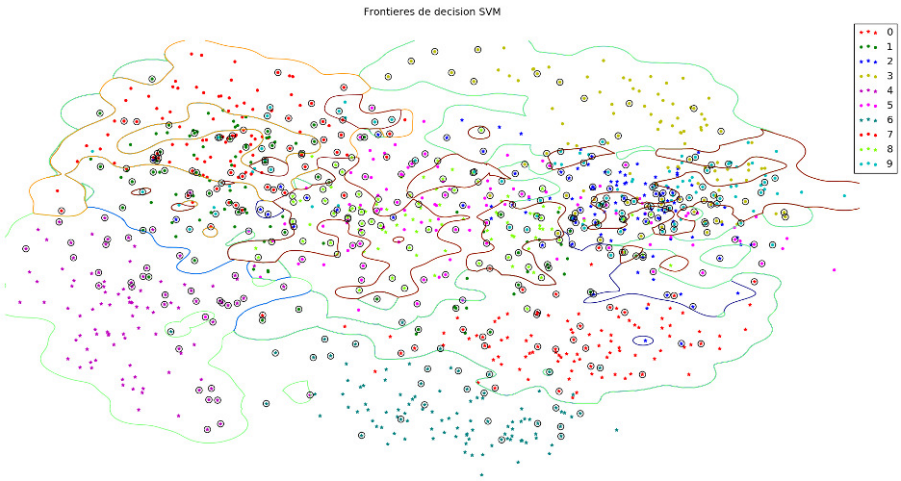
L'utilisation de fonctions de noyau plus complexe, comme une fonction polynomiale, tend à engendrer de *l'overfitting.* Comme dans le cas d'une fonction linéaire, l'entrainement mène à un

résultat de 100% sur le jeu d'entrainement. Le score (*accuracy*) avec le jeu de données de validation est cependant légèrement inférieur. Celui-ci est de 96,9% avec un degré de 2 (alors qu'un score de 97,2% est obtenu avec un kernel linéaire) et se dégrade encore avec des degrés supérieurs.

Lorsqu'on entreprend de procéder à une classification « One versus rest » avec LibSVM, celui-ci tente de représenter $n-1$ ensembles (ou n est le nombre de classes) et considère que l'ensemble restant constitue la dernière classe de données. Cette caractéristique réalise des ensembles dont les frontières épousent plus fidèlement les données d'une classe. Dans notre cas, cela engendre une variance importante du modèle où le dernier ensemble (ici la classe 8) occupe tout l'espace restant comme le montre l'image suivante.



SVM, frontières de décision pour les chiffres 2, 5 et 8 et vecteurs de support avec une machine à vecteurs de support polynomiale de degré 3.



SVM, frontières de décision et vecteurs de support avec une machine à vecteurs de support polynomiale de degré 3.

F. Comparaison des méthodes

L'examen des résultats obtenus avec le *dataset « digits »* révèle que les meilleurs résultats sur le jeu de validation sont obtenus, dans le cas d'espèce, avec les réseaux neuronaux.

De manière indifférenciée sur les trois *datasets,* c'est, par contre, avec les SVM que le modèle affiche les résultats les plus précis.

Type de classification	Score d'entrainement	Score de validation	Score de test
Régression Logistique avec Scikit-Learn et une minimisation par gradient descendant	100%	92%	89%
Régression Logistique avec Theano et une minimisation par gradient descendant stochastique	100%	95,8%	88,7%
Machine à vecteur de support avec un *kernel* linéaire	100%	97,2%	93,1%

Réseau neuronal à couche cachée unique de 500 unités	99,1%	97,9%	91,4%

Tableau comparatif des résultats

Ces résultats démontrent que les propriétés particulières des méthodes de SVM et de réseaux neuronaux permettent de modéliser des domaines d'applications plus complexes en généralisant mieux.

Les résultats de l'algorithme SVM avec *liblinear* sur le *dataset* « *digits* » s'expliquent par le fait qu'il existe un hyperplan optimal qui sépare les données sans devoir augmenter le nombre de dimensions comme on l'a déjà mis en évidence avec les deux tentatives de régression logistique. A la différence des régressions logistiques réalisées, le fait que celui-ci veille à définir la marge la plus vaste entre les classes de données explique le faible taux d'erreur avec des données non rencontrées à l'entrainement.

Il est intéressant de remarquer que le score n'atteint pas 100% lors de l'entrainement d'un réseau neuronal mais que les résultats sont bien meilleurs avec le jeu de validation. Cela met en évidence la faculté à généraliser efficacement inhérente aux réseaux neuronaux.

Conclusion

Le présent mémoire est centré sur le *Machine learning*, un domaine de recherche fascinant et prometteur que j'ai pu découvrir. Cette discipline développe des démarches d'analyse encadrées par un ensemble de bonnes pratiques qui reposent sur des théories mathématiques rigoureuses visant résoudre des problèmes complexes. Aussi ce rapport a-t-il l'ambition de rendre compte des cadres théorique et méthodologique de la discipline et de se livrer ensuite pour l'exemple à l'exploitation d'un jeu de données. Cela a permis d'illustrer le potentiel des techniques issues de ce domaine et mis en lumière que ces techniques d'apprentissage automatique sont particulièrement adaptées et efficaces pour la constitution d'algorithmes décisionnels.

La réussite de la constitution de modèles puissants est conditionnée à la collecte d'un volume suffisant de données pertinentes. Entrainés par de nombreuses données issues de différents domaines d'application, les algorithmes présentés ont conduit à la définition de modèles d'une précision encourageante au moyen de programmes courts. Ces modèles se montrent également évolutifs permettant, à tout moment, d'être entrainés à nouveau avec de nouvelles données.

A titre personnel, l'écriture de ce mémoire m'a permis de comprendre que les mathématiques regorgent de méthodes capables de solutionner des problèmes complexes. Je mesure à présent que l'exploitation de données, par une démarche empirique, constitue une alternative crédible aux approches déterministes.

Au-delà des recherches documentaires dédiées au *Machine Learning*, la rédaction du mémoire m'a poussé, par ailleurs, à m'initier au langage Python et à revoir les bases du calcul vectoriel et matriciel.

Perspectives

Le travail de recherche sur l'apprentissage automatique est probablement à son plus haut niveau d'intensité. Les publications se multiplient et de nouvelles contributions enrichissent chaque mois les connaissances dans ce domaine.

De nombreux produits voient le jour à l'initiative de sociétés privées ou de communautés libres. Leur évaluation pourrait constituer une prolongation pertinente de ce travail.

Des domaines plus spécifiques, tels que l'apprentissage profond (*deep learning*) ou les algorithmes d'apprentissage non-supervisés, pourraient également faire l'objet de développements dédiés.

D'autres méthodes, comme les arbres de décisions, les systèmes de recommandation, les réseaux convolutifs ou l'apprentissage à la volée (*online learning*), auraient encore pu enrichir la sélection d'algorithmes présentés dans le cadre de ce mémoire.

Tout en restant dans l'ensemble de concepts liés au Big data mais en s'éloignant du *Machine learning*, les méthodes de calculs distribués, l'optimisation de l'utilisation des ressources informatiques par *Map reduce* et les questions de prospection de données (*data mining*) pourraient encore représenter des sujets potentiels intéressants.

Voilà autant de pistes qui pourraient utilement prolonger ce mémoire.

Bibliographie

[BANKO 2001] Michele Banko et Eric Brill, *Scaling to Very Very Large Corpora for Natural Language Disambiguation*, http://ucrel.lancs.ac.uk/acl/P/P01/P01-1005.pdf, 2001, consulté le 16 février 2016

[BOTTOU 2012] Bottou, L., *Stochastic Gradient Descent Tricks*, http://research.microsoft.com/pubs/192769/tricks-2012.pdf, 1-9 p., 2012, consulté le 10 avril 2016

[CHANG 2011] Chang Chih-Chung et Lin Chih-Jen, *LIBSVM : a library for support vector machines*, ACM Transactions on Intelligent Systems and Technology, 2:27:1--27:27, 2011

[CHOROMANSKA 2015] Choromanska Anna, Hena Mikael, Mathieu Michael, Ben Arous Gérard, Le Cun Yann, *The Loss Surfaces of Multilayer Networks*, http://arxiv.org/pdf/1412.0233.pdf, 21 janvier 2015, consulté le 17 mars 2016

[DE MONTJOYE 2015] de Montjoye Yves-Alexandre, *« Unique in the shopping mall: On the re-identifiability of credit card metadata »*, Science 347, 30 janvier 2015

[DENIS] Denis F. et Gilleron R., *Apprentissage à partir d'exemples*, http://www.grappa.univ-lille3.fr/polys/apprentissage/sortie005.html, date non trouvée, consulté le 16 mars 2016

[GOUTTE 2005] Goutte C., Gaussier E., *A Probabilistic Interpretation of Precision, Recall and F-score, with Implication for Evaluation*, http://www3.xrce.xerox.com/content/download/20797/148382/file/xrce_eval.pdf, 2005, consulté la 10 avril 2016

[HASTIE 2008] Hastie T., Tibshirani R. et Friedman J., *The Elements of Statistical Learning*, 2e édition, New York : Springer-Verlag, 2008, 37-38 p. (Model Selection and the Bias–Variance Tradeoff)

[HASTIE 2015] Hastie T., Tibshirani R., Wainwright M., *The statistical Learning with Sparsity*, 1e édition, Chapman and Hall, 2015, 1-24 p. (1. Introduction, 2.The Lasso for Linear Models)

[JAMES 2015b] James G., Witten D., Hastie T., Tibshirani R., *An Introduction to Statistical Learning*, 2e édition, New York : Springer, 2015, 33-36 p. (2.2.2 The Bias-Variance Trade-Off)

[JAMES 2015c] James G., Witten D., Hastie T., Tibshirani R., *An Introduction to Statistical Learning*, 2e édition, New York : Springer, 2015, 61-90 p. (3.1 Simple Linear Regression)

[JAMES 2015d] James G., Witten D., Hastie T., Tibshirani R., *An Introduction to Statistical Learning*, 2e édition, New York : Springer, 2015, 104 p.

[JAMES 2015e] James G., Witten D., Hastie T., Tibshirani R., *An Introduction to Statistical Learning*, 2e édition, New York : Springer, 2015, 90-95, 266, 277-280 p.

[JAMES 2015f] James G., Witten D., Hastie T., Tibshirani R., *An Introduction to Statistical Learning*, 2e édition, New York : Springer, 2015, 61-71 p. (3.2 Multiple Linear Regression)

[JAMES 2015g] James G., Witten D., Hastie T., Tibshirani R., *An Introduction to Statistical Learning*, 2e édition, New York : Springer, 2015, 127-150 p. (4. Classification)

[JAMES 2015] James G., Witten D., Hastie T., Tibshirani R., *An Introduction to Statistical Learning*, 2e édition, New York : Springer, 2015, 15-18 p. (2.1 What Is Statistical Learning?)

[KRIS 2013] Kris Mark G., *Memorial Sloan Kettering's Mark Kris on Partnership with IBM Watson*, https://www.mskcc.org/blog/msk-mark-kris-partnership-ibm-watson, 8 février 2013, consulté le 10 janvier 2016

[LEMBERGER 2014] Lemberger Pirmin, *Le « machine learning » – quand les données remplacent les algorithmes*, http://www.journaldunet.com/solutions/expert/56923/le---machine-learning-----quand-les-donnees-remplacent-les-algorithmes.shtml, 28 mars 2014, consulté le 10 janvier 2016

[LENET 2016] Site Web du projet collaboratif Theano, *Documentation officielle*, http://deeplearning.net/tutorial/lenet.html, 2016, consulté le 18 avril

[MARTIN 2016] Martin Charles H., *Why does Deep Learning work?*, https://charlesmartin14.wordpress.com/2015/03/25/why-does-deep-learning-work/, 25 mars 2015, consulté le 17 mars 2016

[MATH_TOULOUSE 2014] Département de Mathématiques de l'université de Toulouse, *Machine à vecteurs de supports*, http://www.math.univ-toulouse.fr/~besse/Wikistat/pdf/st-m-app-svm.pdf, p. 1-16, 19 avril 2014, consulté le 12 avril 2016

[MEEDEN] Meeden Lisa, *Derivation of Backpropagation*, https://www.cs.swarthmore.edu/~meeden/cs81/f15/BackPropDeriv.pdf, date inconnue, consulté le 10 février 2016

[NG 2015a] Ng Andrew, *CS229 Lecture notes : Supervised learning*, http://cs229.stanford.edu/notes/cs229-notes1.pdf, 2015, consulté le 15 octobre 2016

[NG 2015b] Ng Andrew, *CS229 Lecture notes : Regularization and model selection (Cross validation)*, http://cs229.stanford.edu/notes/cs229-notes5.pdf, 2015, consulté le 21 octobre 2016

[NG 2015c] Ng Andrew, *Part XI Principal components analysis*, http://cs229.stanford.edu/notes/cs229-notes10.pdf, 2015, consulté le 20 avril

[NG 2015] Ng Andrew, *CS229 Lecture notes : Supervised learning*, http://cs229.stanford.edu/notes/cs229-notes1.pdf, 2015, consulté le 15 octobre 2016

[NILSSON 1998] Nilsson Nils J., *Introduction to Machine Learning*, Stanford University : Department of Computer Science, 1998, 1-3 p. (Introduction)

[PEDREGOSA 2011] Pedregosa & al., *Scikit-Learn: Machine Learning in Python*, JMLR 12, pp. 2825-2830, 2011

[POWDERS 2011] Powders D. M. W., *Evaluation : From Precision, Recall and F-Measure to ROC Informedness, Markness & Correlation*, http://www.bioinfopublication.org/files/articles/2_1_1_JMLT.pdf, 2011, consulté le 9 avril 2016

[QUOC 2011] Quoc V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, A. Ng, *On optimization methods for deep learning*, http://cs.stanford.edu/~acoates/papers/LeNgiCoaLahProNg11.pdf, 2011, consulté le 10 avril 2016

[ROUVROY 2016] Rouvroy Antoinette, *« Of Data and Men », Fundamental Rights and Freedoms in a World of Big Data*, Council of Europe, Directorate General of Human Rights and Rule of Law, 2016

[SEJNOWSKI 1987] Sejnowski Terrence J. et Rosenberg Charles R., *NETtalk: a parallel network that learns to read aloud*, http://cs.union.edu/~rieffelj/classes/2009-10/csc320/readings/Sejnowski-speech-1987.pdf, 1987, consulté le 17 mars 2016

[SHALEV-SCHWARTZ 2014] Shai Shalev-Shwartz et Shai Ben-David, *Understanding Machine Learning: From Theory to Algorithms*, Cambridge University Press, 2014, 365-368 p. (Feature Manipulation and Normalization)

[SKLEARN a] Communauté de Scikit-Learn, *Documentation officielle de la version 0.17.1*, http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html, 2015, consulté le 4 avril 2016

[SkLEARN b] Communauté de Scikit-Learn, *Documentation officielle de la version 0.17.1*, http://scikit-learn.org/stable/auto_examples/linear_model/plot_iris_logistic.html, 2015, consulté le 5 avril 2016

[SKLEARN c] Communauté de Scikit-Learn, *Documentation officielle de la version 0.17.1*, http://scikit-learn.org/dev/auto_examples/neural_networks/plot_mnist_filters.html, date inconnue, consulté le 16 avril 2016

[SKLEARN d] Communauté de Scikit-Learn, *Documentation officielle de la version 0.18.dev0*, http://scikit-learn.org/dev/modules/generated/sklearn.neural_network.MLPClassifier.html, 2016, consulté le 18 avril 2016

[SKLEARN e] Communauté de Scikit-Learn, *Documentation officielle de la version 0.17.1*, http://scikit-learn.org/stable/auto_examples/svm/plot_iris.html, 2014, consulté le 20 avril

[THEANO 2010] Bastien F., Lamblin P., Pascanu R., Bergstra J., Goodfellow I., Bergeron A., Bouchard N., Warde-Farley D. et Bengio Y.. *"Theano: new features and speed improvements"*, NIPS 2012 deep learning workshop, 2012 ; Bergstra J., Breuleux O., Bastien F., Lamblin P., Pascanu R., Desjardins G., Turian J., Warde-Farley D. et Bengio Y.. *"Theano: A CPU and GPU Math Expression Compiler"*, Proceedings of the Python for Scientific Computing Conference (SciPy) 2010, Austin TX, 30 juin au 3 juillet 2010

[THEANO 2016] Site Web du projet collaboratif Theano, *Documentation officielle*, http://deeplearning.net/software/theano/, 2016, consulté le 15 avril

Annexes

G. Traduction du gradient descendant en Python

Dans le langage python, dans le cadre d'une régression, cet algorithme se traduit comme suit[18] :

```
def hypothesis(X, theta):
    return X.dot(theta)

def computeCost(X, y, theta):
    m = len(y)
    term = hypothesis(X, theta) – y
    return (term.T.dot(term) / (2 * m))[0, 0]

def gradientDescent(X, y, theta, alpha, iterations):
    grad = copy(theta)
    m    = len(y)
    for counter in range(0, iterations):
        inner_sum = X.T.dot(hypothesis(X, grad) - y)
        grad -= alpha / m * inner_sum
    return grad

def compute_regression():

    X, y = « les données d'entrées sous forme de liste où (X(0),y(0)) représente la première
donnée»
    m = len(y)
    X = c_[ones((m, 1)), X]
    theta        = zeros((2, 1))
    iterations = 1500
    alpha = 0.01
    cost         = computeCost(X, y, theta)
    theta = gradientDescent(X, y, theta, alpha, iterations)
    return theta
```

[18] Extraits du fichier source ex1.py mis à disposition librement à l'adresse https://github.com/subokita/mlclass

H. Régression non linéaire (polyniomiale) et évaluation du degré idéal

```
# Regression non lineaire polynomiale de calcul du revenu moyen par habitant (Dataset mis librement a disposition par l'universite de Stanford)
# ecrit au depart du script mis a disposition ici :
#http://stackoverflow.com/questions/19791581/how-to-use-leastsq-function-from-scipy-optimize-in-python-to-fit-both-a-straight

### Import des modules
import sys
# pour le calcul scientifique d'operation mathematiques
from numpy import *
import scipy
from scipy.optimize import leastsq
import numpy as np

# pour l'affichage
from matplotlib import pyplot, cm
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt

EX_DIRECTORY_PATH = 'C:\\Users\\Adrien\\Desktop\\Machine Learning\\mlclass-master\\'

# Fonction d'agencement des donnees en matrice et affichage de celles-ci en deux dimensions
def plot_data():
    data = genfromtxt( EX_DIRECTORY_PATH + "IDH-Esperance de vue-Revenu moyen-TrainingSet.txt", delimiter=',')
    Xtrain, ytrain = data[:, 0], data[:, 1]
    m    = len(ytrain)
    ytrain   = ytrain.reshape(m, 1)
    data = genfromtxt( EX_DIRECTORY_PATH + "IDH-Esperance de vue-Revenu moyen-CrossValidationSet.txt", delimiter=',')
    Xcv, ycv = data[:, 0], data[:, 1]
    m    = len(ycv)
    ycv    = ycv.reshape(m, 1)
    data = genfromtxt( EX_DIRECTORY_PATH + "IDH-Esperance de vue-Revenu moyen-TestSet.txt", delimiter=',')
    Xtest, ytest = data[:, 0], data[:, 1]
    m    = len(ytest)
    ytest    = ytest.reshape(m, 1)
    pyplot.plot(Xtrain, ytrain, 'rx', Xcv, ycv, 'bx', Xtest, ytest, 'gx')
    pyplot.ylabel('Revenu (en $)')
    pyplot.xlabel('Esperance de vie')
    pyplot.show(block=True)

# Fonction de calcul de cout (s'ajuste en fonction du degre polynomial)
def compute_cost(X, y, theta):
    degree = len(theta)-1
    if degree == 1:
```

```
43          hypothesis=lambda tpl,x : tpl[1]*x+tpl[0]
    elif degree == 2:
45          hypothesis=lambda tpl,x : tpl[2]+tpl[1]*x+tpl[0]*x**2
# ...
47      elif degree == 10:
            hypothesis=lambda tpl,x : tpl[10]+tpl[9]*x+tpl[8]*x**2+tpl
        [7]*x**3+tpl[6]*x**4+tpl[5]*x**5+tpl[4]*x**6+tpl[3]*x**7+tpl[2]*
        x**8+tpl[1]*x**9+tpl[0]*x**10
49
        m   = len(y)
51      term = hypothesis(theta, X) - y
        term = term.reshape(m, 1)
53      return (term.T.dot(term) / (2 * m))[0, 0]

55  # Fonction principale
    def main():
57      data = genfromtxt( EX_DIRECTORY_PATH + "IDH-Esperance de vue-
        Revenu moyen-TrainingSet.txt", delimiter=',')
        Xtrain, ytrain = data[:, 0], data[:, 1]
59      data = genfromtxt( EX_DIRECTORY_PATH + "IDH-Esperance de vue-
        Revenu moyen-CrossValidationSet.txt", delimiter=',')
        Xcv, ycv = data[:, 0], data[:, 1]
61      data = genfromtxt( EX_DIRECTORY_PATH + "IDH-Esperance de vue-
        Revenu moyen-TestSet.txt", delimiter=',')
        Xtest, ytest = data[:, 0], data[:, 1]
63
        x=Xtrain
65      y=ytrain

67      funcLine=lambda tpl,x : tpl[1]*x+tpl[0]
        funcPolyDeg2=lambda tpl,x : tpl[2]+tpl[1]*x+tpl[0]*x**2
69  # ...
        funcPolyDeg10=lambda tpl,x : tpl[10]+tpl[9]*x+tpl[8]*x**2+tpl
        [7]*x**3+tpl[6]*x**4+tpl[5]*x**5+tpl[4]*x**6+tpl[3]*x**7+tpl[2]*
        x**8+tpl[1]*x**9+tpl[0]*x**10
71
        ErrorFunc=lambda tpl,x,y: func(tpl,x)-y
73
        ctrain = []
75      ccv = []
        ctest = []
77
        func=funcLine
79      tplInitial1=(0,0)
        tplFinal1,success=leastsq(ErrorFunc,tplInitial1[:],args=(x,y))
81      print("Representation lineaire",tplFinal1)
        xx1=np.linspace(x.min(),x.max(),86)
83      yy1=func(tplFinal1,xx1)
        ctrain1=compute_cost(x, y, tplFinal1)
85      ccv1=compute_cost(Xcv, ycv, tplFinal1)
        ctest1=compute_cost(Xtest, ytest, tplFinal1)
87      print('Cout TrainingSet = ',ctrain1)
        print('Cout CrossValidationSet = ',ccv1)
```

```
89      print('Cout TestSet = ',ctest1)
        ctrain.append(ctrain1)
91      ccv.append(ccv1)
        ctest.append(ctest1)
93
        func=funcPolyDeg2
95      tplInitial2=(0,0,0)
        tplFinal2,success=leastsq(ErrorFunc,tplInitial2[:],args=(x,y))
97      print("Representation polynomiale de degre 2" ,tplFinal2)
        xx2=xx1
99      yy2=func(tplFinal2,xx2)
        ctrain2=compute_cost(x, y, tplFinal2)
101     ccv2=compute_cost(Xcv, ycv, tplFinal2)
        ctest2=compute_cost(Xtest, ytest, tplFinal2)
103     print('Cout TrainingSet = ',ctrain2)
        print('Cout CrossValidationSet = ',ccv2)
105     print('Cout TestSet = ',ctest2)
        ctrain.append(ctrain2)
107     ccv.append(ccv2)
        ctest.append(ctest2)
109
    # ...
111
        func=funcPolyDeg10
113     tplInitial10=(0,0,0,0,0,0,0,0,0,0,0)
        tplFinal10,success=leastsq(ErrorFunc,tplInitial10[:],args=(x,y)
        )
115     print("Representation polynomiale de degre 10" ,tplFinal10)
        xx10=xx1
117     yy10=func(tplFinal10,xx10)
        ctrain10=compute_cost(x, y, tplFinal10)
119     ccv10=compute_cost(Xcv, ycv, tplFinal10)
        ctest10=compute_cost(Xtest, ytest, tplFinal10)
121     print('Cout TrainingSet = ',ctrain10)
        print('Cout CrossValidationSet = ',ccv10)
123     print('Cout TestSet = ',ctest10)
        ctrain.append(ctrain10)
125     ccv.append(ccv10)
        ctest.append(ctest10)
127
        plt.plot(xx1,yy1,'b-', label="Polynome de degre 1")
129     plt.plot(xx2,yy2,'g-', label="Polynome de degre 2")
    # ...
131     plt.plot(xx10,yy10,'k-', label="Polynome de degre 10")
        plt.plot(x,y,'rx', label="Training set")
133     plt.plot(Xcv, ycv, 'bx', label="Cross Validation set")
        plt.plot(Xtest, ytest, 'gx', label="Test set")
135     plt.legend(bbox_to_anchor=(0.5, 0.95), loc=1, borderaxespad
        =-0.2)
        pyplot.ylabel('Revenu (en $)')
137     pyplot.xlabel('Esperance de vie')
        plt.show()
139
```

```
    print('Cout entrainement',ctrain)
    print('Cout validation',ccv)
    print('Cout test',ctest)
    degree=np.linspace(1,20,20)
    print(degree)

# Affichage des graphiques
    plt.plot(degree,ctrain,'r-', label="Training set")
    plt.legend(bbox_to_anchor=(0.5, 0.95), loc=1, borderaxespad
    =-0.2)
    pyplot.ylabel('Cout')
    pyplot.xlabel('Degre polynomial')
    plt.show(block=False)

    plt.plot(degree,ccv,'b-', label="Cross Validation set")
    plt.legend(bbox_to_anchor=(0.5, 0.95), loc=1, borderaxespad
    =-0.2)
    pyplot.ylabel('Cout')
    pyplot.xlabel('Degre polynomial')
    plt.show(block=True)

    plt.plot(degree,ctest,'g-', label="Test set")
    plt.plot(degree,ctest,'b-', label="Cross Validation set")
    plt.legend(bbox_to_anchor=(0.5, 0.95), loc=1, borderaxespad
    =-0.2)
    pyplot.ylabel('Cout')
    pyplot.xlabel('Degre polynomial')
    plt.show(block=True)

if __name__=="__main__":
    main()
```

I. Régression logistique sur le jeu de données Digits avec Scikit-learn

```
# Regression logistique (Dataset Digits issus de Scikit-Learn)
# ecrit au depart de la documentation mise a disposition ici :
#http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
# Le premier arguments est le learning rate (symbole decimal = .).
# Le second argument le nombre maximal d'iterations
# Le troisieme argument est le type d'algorithme utilise pour la minimisation.
# Le quatrieme argument determine si on laisse ou non l'algorithme normaliser les variables d'entree

### Import des modules
from sklearn import datasets # pour le chargement du dataset
import pylab as pl # pour l'affichage d'une image
import numpy as np
from sklearn import linear_model # pour realiser une regression logistique
import sys # pour le logging
import timeit

# Recuperation des arguments d'appel de ce script
LEARNING_RATE = float(sys.argv[1])
N_EPOCHS = int(sys.argv[2])
SOLVER = sys.argv[3] # 'newton-cg', 'lbfgs', 'liblinear', 'sag'
#RANDOM_STATE = sys.argv[4] # int seed, RandomState instance, or None (default)
INTERCEPT_SCALING = sys.argv[4] # True or False (only with LibLinear)
print("%f;%d;%s" % (LEARNING_RATE,N_EPOCHS,SOLVER))

    ############################
    # Chargement des donnees #
    ############################

digits = datasets.load_digits()
digits.images.shape

### Contenu de digits ###
# {'images': array([[[  0.,   0.,   5., ...,   1.,   0.,   0.],
        # [  0.,   0.,  13., ...,  15.,   5.,   0.],
        # [  0.,   3.,  15., ...,  11.,   8.,   0.],
        # ...,
        # [  0.,   0.,   1., ...,   6.,   0.,   0.],
        # [  0.,   0.,   2., ...,  12.,   0.,   0.],
        # [  0.,   0.,  10., ...,  12.,   1.,   0.]]), '
    target_names': array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),..., '
    target': array([0, 1, 2, ..., 8, 9, 8])}

### Vectorisation d'une image en un vecteur de 64 features
X_data = digits.images.reshape((digits.images.shape[0], -1))
# >>> data.shape
# (1797L, 64L)
```

```
45
### Normalization
47 X_data = (X_data - np.min(X_data, 0)) / (np.max(X_data, 0) +
    0.0001)  # Mise a l'echelle des donnees entree dans une plage de
     valeur comprise entre 0 et 1

49 y_data = digits.target
# >>> y_data
51 # array([0, 1, 2, ..., 8, 9, 8])

53 n_samples = len(X_data)
# >>> len(data)
55 # 1797

57     ##############################
       # Fragmentation du dataset #
59     ##############################

61 X_train = X_data[:int(0.6 * n_samples)] # 60 premiers % de la
    liste
   y_train = y_data[:int(0.6 * n_samples)]
63 X_temp = X_data[int(0.6 * n_samples):] # 40 derniers % de la liste
   y_temp = y_data[int(0.6 * n_samples):]
65 X_test = X_data[int(0.8 * n_samples):] # 20 derniers % de la liste
   y_test = y_data[int(0.8 * n_samples):]
67 X_cv = X_temp[:int(0.5 * n_samples)] # enrigistrements situes de
    60% a 80% de la liste
   y_cv = y_temp[:int(0.5 * n_samples)]
69
   start_time = timeit.default_timer()
71
       ############################
73     # Construction du modele #
       ############################
75
   logistic = linear_model.LogisticRegression(max_iter=N_EPOCHS, C=
    LEARNING_RATE, solver=SOLVER)
77
       ############################
79     # Entrainement du modele #
       ############################
81
   model = logistic.fit(X_train,y_train)
83
   end_time = timeit.default_timer()
85 TRAINING_TIME = end_time - start_time

87     ##############################
       # Affichage des resultats #
89     ##############################

91 print('Taux d\'apprentissage (C);Algorithme de minimisation;Nombre
    d\'iterations;Score d\'entrainement;Score de validation;Score
```

```
    de test;Temps ecoule')
print("Optimization complete;%.3f;%s;%.0f;%.3f;%.3f;%.3f;%.1f" % (
    LEARNING_RATE,SOLVER,N_EPOCHS,model.score(X_train,y_train),model
    .score(X_cv,y_cv),model.score(X_test,y_test),TRAINING_TIME))
```

J. Régression logistique sur le jeu de données Digits avec Theano

```
# Regression logistique utilisant l'algorithme du gradient
   descendant stochastique realise avec Theano 0.8.2 (Dataset
   Digits issu de Scikit-Learn)
# ecrit au depart de la documentation mise a disposition ici :
#http://www.deeplearning.net/tutorial/logreg.html#logreg

### Import des modules
import six.moves.cPickle as pickle
import gzip
import os
import sys
import timeit
import numpy as np
import pylab as pl
from sklearn import datasets
import theano
import theano.tensor as T

# Classe permettant la creation d'un objet representant une
   Regression Logistique comportant differents attributs tels que
   la matrice de poids, l'unite bios, etc.
class LogisticRegression(object):

# Constructeur de l'objet qui initialise les attributs de celui-ci
   .
#  Recoit en argument une variable symbolique qui decrit le type d
   'architecture, le nombre d'entree et le nombre de sorties
    def __init__(self, input, n_in, n_out):

# initialition de la matrice de poids de l'objet a 0
        self.W = theano.shared(
            value=np.zeros(
                (n_in, n_out),
                dtype=theano.config.floatX
            ),
            name='W',
            borrow=True
        )
# initialisation du poids de l'unites Bios
        self.b = theano.shared(
            value=np.zeros(
                (n_out,),
                dtype=theano.config.floatX
            ),
            name='b',
            borrow=True
        )

# Expression symbolique de la fonction de classification (Theano
   la compile avant toute utilisation)
        self.p_y_given_x = T.nnet.softmax(T.dot(input, self.W) +
   self.b)
```

```
# Expression symbolique de la fonction de prediction (Theano la
    compile avant toute utilisation)
        self.y_pred = T.argmax(self.p_y_given_x, axis=1)

# Integration de l'ensemble des parametres dans un vecteur unique
    (Matrice de poids et poids de l'unite Bios)
        self.params = [self.W, self.b]

        self.input = input

# Fonction utile au calcul de cout (la somme des poids est ici
    remplacee par sa moyenne pour rendre le learning rate moins
    dependant de la taille du batch)
#  y est la valeur connue de la classe pour les donnees d'entree
    def negative_log_likelihood(self, y):
        return -T.mean(T.log(self.p_y_given_x)[T.arange(y.shape
    [0]), y])

# Fonction retournant le nombre d'erreurs de classification
    rencontrees sur le nombre d'exemples present dans celui-ci
    def errors(self, y):
        if y.ndim != self.y_pred.ndim:
            raise TypeError(
                'y should have the same shape as self.y_pred',
                ('y', y.type, 'y_pred', self.y_pred.type)
            )

      if y.dtype.startswith('int'):
            return T.mean(T.neq(self.y_pred, y))
        else:
            raise NotImplementedError()

# Fonction chargeant le dataset
def load_data():

    ##############
    # LOAD DATA #
    ##############
    print('... loading data')

    digits = datasets.load_digits()
    X_data = digits.images.reshape(digits.images.shape[0], -1)
    y_data = y_data = digits.target

# Segmentation du dataset en trois jeux de donnees (entrainement,
    validation, test)
    n_samples = len(X_data)
    p60=int(0.6 * n_samples)
    train_set_x = X_data[:p60] # 60 premiers % de la liste
    train_set_y = y_data[:p60]
    train_set = train_set_x, train_set_y
    X_temp = X_data[p60:] # 40 derniers % de la liste
```

```
    y_temp = y_data[p60:]
    n_temp_samples = len(X_temp)
    p50=int(0.5 * n_temp_samples)
    valid_set_x = X_temp[:p50] # enrigistrements situes de 60% a 80% de la liste
    valid_set_y = y_temp[:p50]
    valid_set = valid_set_x, valid_set_y
    test_set_x = X_temp[p50:] # 20 derniers % de la liste
    test_set_y = y_temp[p50:]
    test_set = test_set_x, test_set_y

# Inner class utilisee pour le chargement des donnees comme 'Variables partagees' (Cela permet par exemple d'executer ce code sur un GPU)
    def shared_dataset(data_xy, borrow=True):

        data_x, data_y = data_xy
        shared_x = theano.shared(np.asarray(data_x,dtype=theano.config.floatX),borrow=borrow)
        shared_y = theano.shared(np.asarray(data_y,dtype=theano.config.floatX),borrow=borrow)
        return shared_x, T.cast(shared_y, 'int32')

    test_set_x, test_set_y = shared_dataset(test_set)
    valid_set_x, valid_set_y = shared_dataset(valid_set)
    train_set_x, train_set_y = shared_dataset(train_set)

    rval = [(train_set_x, train_set_y), (valid_set_x, valid_set_y),
            (test_set_x, test_set_y)]
    return rval

# Fonction d'optimisation par gradient descendant mini-batch
def sgd_optimization(learning_rate=0.13, n_epochs=1000,dataset='', batch_size=60):

# Chargement di dataset
    datasets = load_data()

    train_set_x, train_set_y = datasets[0]
    valid_set_x, valid_set_y = datasets[1]
    test_set_x, test_set_y = datasets[2]

# Calcul du nombre de mini-batch pour les phases d'entrainement, de validation et de test
    n_train_batches = train_set_x.get_value(borrow=True).shape[0] // batch_size
    n_valid_batches = valid_set_x.get_value(borrow=True).shape[0] // batch_size
    n_test_batches = test_set_x.get_value(borrow=True).shape[0] // batch_size

    ############################
```

```
    # Construction du modele #
134 ##########################
    print('... building the model')
136
    index = T.lscalar()  # variable symbolique d'index des lots de
    donnees mini-batch
138
    # Variables symboliques pour les entrees
140 x = T.matrix('x')  # donnees
    y = T.ivector('y')  # classifications connues
142
    # Construction de l'objet classifier de type
   LogisticRegression
144 classifier = LogisticRegression(input=x, n_in=8 * 8, n_out=10)

146 # Fonction de cout utile a la minimisation
    cost = classifier.negative_log_likelihood(y)
148
  # Compilation de la fonction de calcul des erreurs de
   classification
150 test_model = theano.function(
        inputs=[index],
152     outputs=classifier.errors(y),
        givens={
154         x: test_set_x[index * batch_size: (index + 1) *
   batch_size],
            y: test_set_y[index * batch_size: (index + 1) *
   batch_size]
156     }
    )
158
    validate_model = theano.function(
160     inputs=[index],
        outputs=classifier.errors(y),
162     givens={
            x: valid_set_x[index * batch_size: (index + 1) *
   batch_size],
164         y: valid_set_y[index * batch_size: (index + 1) *
   batch_size]
        }
166 )

168 # Calcul de l'algorithme du gradient avec la matrice de cout
   et l'unite Bios
    g_W = T.grad(cost=cost, wrt=classifier.W)
170 g_b = T.grad(cost=cost, wrt=classifier.b)

172 # Specification de la maniere dont les variables d'entree sont
    modifiees
    updates = [(classifier.W, classifier.W - learning_rate * g_W),
174            (classifier.b, classifier.b - learning_rate * g_b)]

176 # Compilation de la fonction d'entrainement retournant le cout
```

```
    et les ajustements des variables
    train_model = theano.function(
178         inputs=[index],
            outputs=cost,
180         updates=updates,
            givens={
182             x: train_set_x[index * batch_size: (index + 1) *
    batch_size],
                y: train_set_y[index * batch_size: (index + 1) *
    batch_size]
184         }
    )
186
    ###############################
188 # Entrainement du modele #
    ###############################
190 print('... training the model')
    # Parametres d'arr t de la minimisation
192 patience = 500  # Nombre d'exemples a regarder au minimum
    patience_increase = 2  # Nombre d'iterations minimales a
    realiser apres toute decouverte d'un mailleur cout
194 improvement_threshold = 0.995  # Seuil de progres minimal pour
    considerer qu'il y a un gain
    validation_frequency = min(n_train_batches, patience // 2) #
    Quand calculer le score de validation
196
    best_validation_loss = np.inf
198 test_score = 0.
    start_time = timeit.default_timer()
200
    done_looping = False
202 epoch = 0
    while (epoch < n_epochs) and (not done_looping):
204     epoch = epoch + 1
        for minibatch_index in range(n_train_batches):
206
            minibatch_avg_cost = train_model(minibatch_index)
208         iter = (epoch - 1) * n_train_batches + minibatch_index

210         if (iter + 1) % validation_frequency == 0:
                validation_losses = [validate_model(i)
212                                  for i in range(
    n_valid_batches)]
                this_validation_loss = np.mean(validation_losses)
214
                print(
216                 'epoch %i, minibatch %i/%i, validation error %
    f %%' %
                    (
218                     epoch,
                        minibatch_index + 1,
220                     n_train_batches,
                        this_validation_loss * 100.
```

```
                    )
                )

                if this_validation_loss < best_validation_loss:
                    if this_validation_loss < best_validation_loss
 *  \
                       improvement_threshold:
                        patience = max(patience, iter *
patience_increase)

                    best_validation_loss = this_validation_loss
                    test_losses = [test_model(i)
                                   for i in range(n_test_batches)]
                    test_score = np.mean(test_losses)

                    print(
                        (
                            '     epoch %i, minibatch %i/%i, test
error of'
                            ' best model %f %%'
                        ) %
                        (
                            epoch,
                            minibatch_index + 1,
                            n_train_batches,
                            test_score * 100.
                        )
                    )

                    # Sauvegarde du meilleur modele
                    with open('best_model.pkl', 'wb') as f:
                        pickle.dump(classifier, f)

            if patience <= iter:
                done_looping = True
                break

    end_time = timeit.default_timer()
    print(
        (
            'Optimization complete with best validation score of %
f %%,'
            'with test performance %f %%'
        )
        % (best_validation_loss * 100., test_score * 100.)
    )
    print('The code run for %d epochs, with %f epochs/sec' % (
        epoch, 1. * epoch / (end_time - start_time)))
    print(('The code for file ' +
           os.path.split(__file__)[1] +
           ' ran for %.1fs' % ((end_time - start_time))), file=sys
.stderr)
```

```
270 # Fonction permettant de realiser des predictions au regard d'une
    donnee entree selon le meilleur modele entraine
    def predict():
272
        # Chargement du modele
274     classifier = pickle.load(open('best_model.pkl'))

276     # Compilation de la fonction de prediction
        predict_model = theano.function(
278         inputs=[classifier.input],
            outputs=classifier.y_pred)
280
        # Predictions de dix donnees a titre d'exemple
282     datasets = load_data()
        test_set_x, test_set_y = datasets[2]
284     test_set_x = test_set_x.get_value()

286     predicted_values = predict_model(test_set_x[:10])
        print("Predicted values for the first 10 examples in test set:
    ")
288     print(predicted_values)


290
    if __name__ == '__main__':
292     sgd_optimization()
```

K. Réseau neuronal sur le jeu de données Digits avec Theano

```
# Regression logistique utilisant l'algorithme du gradient
   descendant stochastique realise avec Theano 0.8.2 (Dataset
   Digits issu de Scikit-Learn)
# ecrit au depart de la documentation mise a disposition ici :
#http://www.deeplearning.net/tutorial/mlp.html#mlp

### Import des modules
from __future__ import print_function
import os
import sys
import timeit
import numpy
import theano
import theano.tensor as T

# Chargement de la classe LogisticRegression (depuis le script
   dedie a la regression logistique realisee avec Theano)
from logistic_sgd_Theano_digits_dataset import LogisticRegression,
    load_data

# Declaration de la classe objet HiddenLayer constituant une
   couche et de ses attributs
class HiddenLayer(object):

# Constructeur de la classe
    def __init__(self, rng, input, n_in, n_out, W=None, b=None,
                 activation=T.tanh):
        """
        Typical hidden layer of a MLP: units are fully-connected
   and have
        sigmoidal activation function. Weight matrix W is of shape
    (n_in,n_out)
        and the bias vector b is of shape (n_out,).

        NOTE : The nonlinearity used here is tanh

        Hidden unit activation is given by: tanh(dot(input,W) + b)

        :type rng: numpy.random.RandomState
        :param rng: a random number generator used to initialize
   weights

        :type input: theano.tensor.dmatrix
        :param input: a symbolic tensor of shape (n_examples, n_in
   )

        :type n_in: int
        :param n_in: dimensionality of input

        :type n_out: int
        :param n_out: number of hidden units
```

```
44          :type activation: theano.Op or function
            :param activation: Non linearity to be applied in the
    hidden
46                             layer
            """
48          self.input = input

50  # Initialisation des poids de la couche
            if W is None:
52              W_values = numpy.asarray(
                    rng.uniform(
54                      low=-numpy.sqrt(6. / (n_in + n_out)),
                        high=numpy.sqrt(6. / (n_in + n_out)),
56                      size=(n_in, n_out)
                    ),
58                  dtype=theano.config.floatX
                )
60              if activation == theano.tensor.nnet.sigmoid:
                    W_values *= 4
62
                W = theano.shared(value=W_values, name='W', borrow=
    True)
64
66  # Initialisation du poids de l'unite Bios
            if b is None:
68              b_values = numpy.zeros((n_out,), dtype=theano.config.
    floatX)
                b = theano.shared(value=b_values, name='b', borrow=
    True)

70          self.W = W
            self.b = b
72
            lin_output = T.dot(input, self.W) + self.b
74          self.output = (
                lin_output if activation is None
76              else activation(lin_output)
            )
78          # parameters of the model
            self.params = [self.W, self.b]
80


82  # Declaration de la classe objet MLP
    class MLP(object):
84
    # Constructeur de la classe
86      def __init__(self, rng, input, n_in, n_hidden, n_out):
            """Initialize the parameters for the multilayer perceptron
88
            :type rng: numpy.random.RandomState
90          :param rng: a random number generator used to initialize
    weights
```

```
92          :type input: theano.tensor.TensorType
            :param input: symbolic variable that describes the input
    of the
94          architecture (one minibatch)

96          :type n_in: int
            :param n_in: number of input units, the dimension of the
    space in
98          which the datapoints lie

100         :type n_hidden: int
            :param n_hidden: number of hidden units
102
            :type n_out: int
104         :param n_out: number of output units, the dimension of the
     space in
            which the labels lie
106
            """
108
            # Since we are dealing with a one hidden layer MLP, this
    will translate
110         # into a HiddenLayer with a tanh activation function
    connected to the
            # LogisticRegression layer; the activation function can be
     replaced by
112         # sigmoid or any other nonlinear function
            self.hiddenLayer = HiddenLayer(
114             rng=rng,
                input=input,
116             n_in=n_in,
                n_out=n_hidden,
118             activation=T.tanh
            )
120
            # The logistic regression layer gets as input the hidden
    units
122         # of the hidden layer
            self.logRegressionLayer = LogisticRegression(
124             input=self.hiddenLayer.output,
                n_in=n_hidden,
126             n_out=n_out
            )
128         # L1 norm ; one regularization option is to enforce L1
    norm to
            # be small
130         self.L1 = (
                abs(self.hiddenLayer.W).sum()
132             + abs(self.logRegressionLayer.W).sum()
            )
134
            # square of L2 norm ; one regularization option is to
    enforce
```

```
136         # square of L2 norm to be small
            self.L2_sqr = (
138             (self.hiddenLayer.W ** 2).sum()
                + (self.logRegressionLayer.W ** 2).sum()
140         )

142         # negative log likelihood of the MLP is given by the
    negative
            # log likelihood of the output of the model, computed in
    the
144         # logistic regression layer
            self.negative_log_likelihood = (
146             self.logRegressionLayer.negative_log_likelihood
            )
148         # same holds for the function computing the number of
    errors
            self.errors = self.logRegressionLayer.errors
150
            # the parameters of the model are the parameters of the
    two layer it is
152         # made out of
            self.params = self.hiddenLayer.params + self.
    logRegressionLayer.params
154
            # keep track of model input
156         self.input = input

158 # Fonction d'optimisation par une approche stochastique
    def sgd_optimization_mlp(learning_rate=0.01, L1_reg=0.00, L2_reg
    =0.0001, n_epochs=1000,
160                 dataset='', batch_size=20, n_hidden=500):

162 # Chargement du dataset
        datasets = load_data(dataset)
164
        train_set_x, train_set_y = datasets[0]
166     valid_set_x, valid_set_y = datasets[1]
        test_set_x, test_set_y = datasets[2]
168
    # Calcul du nombre de mini-batch pour les phases d'entrainement,
    de validation et de test
170     n_train_batches = train_set_x.get_value(borrow=True).shape[0]
    // batch_size
        n_valid_batches = valid_set_x.get_value(borrow=True).shape[0]
    // batch_size
172     n_test_batches = test_set_x.get_value(borrow=True).shape[0] //
        batch_size

174     ######################
        # Construction du modele #
176     ######################
        print('... building the model')
178
```

```
    index = T.lscalar()  # variable symbolique d'index des lots de
    donnees mini-batch
180
    # Variables symboliques pour les entrees
182 x = T.matrix('x')  # donnees
    y = T.ivector('y')  # classifications connues
184
  # Initialisation des poids par le biais d'un nombre aleatoire
186 rng = numpy.random.RandomState(1234)

188 # Construction de l'objet classifier de type
   LogisticRegression MLP
    classifier = MLP(
190     rng=rng,
        input=x,
192     n_in=8 * 8,
        n_hidden=n_hidden,
194     n_out=10
    )
196
    # Fonction de cout utile a la minimisation
198 cost = (
        classifier.negative_log_likelihood(y)
200     + L1_reg * classifier.L1
        + L2_reg * classifier.L2_sqr
202 )

204 # Compilation de la fonction de calcul des erreurs de
   classification
    test_model = theano.function(
206     inputs=[index],
        outputs=classifier.errors(y),
208     givens={
            x: test_set_x[index * batch_size:(index + 1) *
   batch_size],
210         y: test_set_y[index * batch_size:(index + 1) *
   batch_size]
        }
212 )

214 validate_model = theano.function(
        inputs=[index],
216     outputs=classifier.errors(y),
        givens={
218         x: valid_set_x[index * batch_size:(index + 1) *
   batch_size],
            y: valid_set_y[index * batch_size:(index + 1) *
   batch_size]
220     }
    )
222
    # Calcul de l'algorithme du gradient avec la matrice de cout
   et l'unite Bios
```

```
    gparams = [T.grad(cost, param) for param in classifier.params]

    # Specification de la maniere dont les variables d'entree sont
    modifiees
    updates = [
        (param, param - learning_rate * gparam)
        for param, gparam in zip(classifier.params, gparams)
    ]

    # Compilation de la fonction d'entrainement retournant le cout
    et les ajustements des variables
    train_model = theano.function(
        inputs=[index],
        outputs=cost,
        updates=updates,
        givens={
            x: train_set_x[index * batch_size: (index + 1) *
batch_size],
            y: train_set_y[index * batch_size: (index + 1) *
batch_size]
        }
    )

    ###############################
    # Entrainement du modele #
    ###############################
    print('... training')

    # Parametres d'arr t de la minimisation
    patience = 500  # Nombre d'exemples a regarder au minimum
    patience_increase = 2  # Nombre d'iterations minimales a
realiser apres toute decouverte d'un mailleur cout
    improvement_threshold = 0.995  # Seuil de progres minimal pour
    considerer qu'il y a un gain
    validation_frequency = min(n_train_batches, patience // 2)#
Quand calculer le score de validation (a l'affichage)

    best_validation_loss = numpy.inf
    best_iter = 0
    test_score = 0.
    start_time = timeit.default_timer()

    epoch = 0
    done_looping = False

    while (epoch < n_epochs) and (not done_looping):
        epoch = epoch + 1
        for minibatch_index in range(n_train_batches):

            minibatch_avg_cost = train_model(minibatch_index)
            # iteration number
            iter = (epoch - 1) * n_train_batches + minibatch_index
```

```
270             if (iter + 1) % validation_frequency == 0:

272         validation_losses = [validate_model(i) for i
                                 in range(n_valid_batches)]
274                 this_validation_loss = numpy.mean(
    validation_losses)

276                 print(
                        'epoch %i, minibatch %i/%i, validation error %
    f %%' %
278                         (
                                epoch,
280                             minibatch_index + 1,
                                n_train_batches,
282                             this_validation_loss * 100.
                            )
284                 )

286                 if this_validation_loss < best_validation_loss:
                        if (
288                             this_validation_loss <
    best_validation_loss *
                                improvement_threshold
290                     ):
                            patience = max(patience, iter *
    patience_increase)
292
                        best_validation_loss = this_validation_loss
294                     best_iter = iter

296                     test_losses = [test_model(i) for i
                                       in range(n_test_batches)]
298                     test_score = numpy.mean(test_losses)

300                     print(('     epoch %i, minibatch %i/%i, test
    error of '
                               'best model %f %%') %
302                              (epoch, minibatch_index + 1,
    n_train_batches,
                               test_score * 100.))
304
            if patience <= iter:
306                 done_looping = True
                    break
308
     end_time = timeit.default_timer()
310  print(('Optimization complete. Best validation score of %f %%
    '
           'obtained at iteration %i, with test performance %f %%'
    ) %
312         (best_validation_loss * 100., best_iter + 1, test_score
    * 100.))
     print(('The code for file ' +
```

```
314             os.path.split(__file__)[1] +
                ' ran for %.2fm' % ((end_time - start_time) / 60.)),
    file=sys.stderr)
316
318 if __name__ == '__main__':
     sgd_optimization_mlp()
```

L. Réseau neuronal sur le jeu de données Digits avec Scikit-Learn

```
# Reseau Neuronal realise avec Scikit-Learn 0.18.dev0 (Dataset Digits issus de Scikit-Learn)
# ecrit au depart de la documentation mise a disposition ici :
#http://scikit-learn.org/dev/modules/generated/sklearn.neural_network.MLPClassifier.html
# Le premier argument est la constitution des couches cachees (nombres d'unites par couche separes par des virgules)
# Le second argument determine la fonction d'activation d'une unite (logistic, tanh, relu)
# Le troisieme argument est le type d'algorithme utilise pour la minimisation (l-bfgs, sgd, adam).
# Le quatrieme argument est un multiplicateur du parametre de regularisation
# Le cinquieme argument est la taille d'un lot de donnees mini-batch (ignore si l-bfgs)
# Le sixieme argument est le learning rate (initial si on choisit de le faire varier)
# Le septieme argument est le mode d'ajustement du learning rate (constant, invscaling, adaptive)

### Import des modules
from sklearn import datasets # pour le chargement du dataset
import pylab as pl # pour l'affichage d'une image
from sklearn.neural_network import MLPClassifier
import sys # pour le logging

# Recuperation des arguments d'appel de ce script
HIDDEN_LAYER_SIZES = tuple(map(int,sys.argv[1].split(",")))
ACTIVATION = sys.argv[2] # 'relu'
ALGORITHM = sys.argv[3] # 'adam'
ALPHA = float(sys.argv[4]) # 0.0001
BATCH_SIZE = sys.argv[5]# 'auto'
LEARNING_RATE = sys.argv[6] # 'constant'
LEARNING_RATE_INIT=0.001
POWER_T=0.5
MAX_ITER=20000
SHUFFLE=True
RANDOM_STATE = 111
TOL=1e-6
MOMENTUM=0.9
NESTEROVS_MOMENTUM=True
EARLY_STOPPING=False
VALIDATION_FRACTION=0.1
BETA_1=0.9
BETA_2=0.999
EPSILON=1e-08

    ############################
    # Chargement des donnees #
    ############################

digits = datasets.load_digits()
```

```
digits.images.shape

### Contenu de digits ###
# {'images': array([[[  0.,   0.,   5., ...,   1.,   0.,   0.],
#         [  0.,   0.,  13., ...,  15.,   5.,   0.],
#         [  0.,   3.,  15., ...,  11.,   8.,   0.],
#         ...,
#         [  0.,   0.,   1., ...,   6.,   0.,   0.],
#         [  0.,   0.,   2., ...,  12.,   0.,   0.],
#         [  0.,   0.,  10., ...,  12.,   1.,   0.]]), '
   target_names': array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),..., '
   target': array([0, 1, 2, ..., 8, 9, 8])}

### Affichage d'une image ###
# pl.gray()
# pl.matshow(digits.images[0])
# pl.show()

### Vectorisation d'une image en un vecteur de 64 features
X_data = digits.images.reshape((digits.images.shape[0], -1))
# >>> data
# array([[  0.,   0.,   5., ...,   0.,   0.,   0.],
#         [  0.,   0.,   0., ...,  10.,   0.,   0.],
#         [  0.,   0.,   0., ...,  16.,   9.,   0.],
#         ...,
#         [  0.,   0.,   1., ...,   6.,   0.,   0.],
#         [  0.,   0.,   2., ...,  12.,   0.,   0.],
#         [  0.,   0.,  10., ...,  12.,   1.,   0.]])
# >>> data.shape
# (1797L, 64L)

y_data = digits.target
# >>> y_data
# array([0, 1, 2, ..., 8, 9, 8])

n_samples = len(X_data)
# >>> len(data)
# 1797

# Segmentation du dataset
X_train = X_data[:int(0.6 * n_samples)] # 60 premiers % de la
   liste
y_train = y_data[:int(0.6 * n_samples)]
X_temp = X_data[int(0.6 * n_samples):] # 40 derniers % de la liste
y_temp = y_data[int(0.6 * n_samples):]
X_test = X_data[int(0.8 * n_samples):] # 20 derniers % de la liste
y_test = y_data[int(0.8 * n_samples):]
X_cv = X_temp[:int(0.5 * n_samples)] # enrigistrements situes de
   60% a 80% de la liste
y_cv = y_temp[:int(0.5 * n_samples)]

# Construction du modele
classifier = MLPClassifier(hidden_layer_sizes=HIDDEN_LAYER_SIZES,
```

```
    activation=ACTIVATION, algorithm=ALGORITHM, alpha=ALPHA,
    batch_size=BATCH_SIZE, learning_rate=LEARNING_RATE,
    learning_rate_init=LEARNING_RATE_INIT, power_t=POWER_T, max_iter
    =MAX_ITER, shuffle=True, random_state=RANDOM_STATE, tol=TOL,
    verbose=False, warm_start=False, momentum=MOMENTUM,
    nesterovs_momentum=NESTEROVS_MOMENTUM, early_stopping=
    EARLY_STOPPING, validation_fraction=VALIDATION_FRACTION, beta_1=
    BETA_1, beta_2=BETA_2, epsilon=EPSILON)

# Entraienement du modele
trained_classifier=classifier.fit(X_train, y_train)

# Affichage des resultats
#print('Nombre d\'iterations ; Score d\'entrainement ; Score de
    validation ; Score de test')
print(('%s;%d;%.3f;%.3f;%.3f') % (str(HIDDEN_LAYER_SIZES),
    trained_classifier.n_iter_,trained_classifier.score(X_train,
    y_train),trained_classifier.score(X_cv, y_cv),trained_classifier
    .score(X_test, y_test)))
```

M. Machine à vecteur de support sur le jeu de données Digits avec Scikit-Learn

```
# Support Vector Machine (Dataset Digits issu de scikit-learn)
# ecrit au depart de la documentation mise a disposition ici :
#http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
# Le premier arguments est le learning rate (symbole decimal = .).
# Le second argument est le type de kernel voulu ( linear , poly , rbf , sigmoid , precomputed ).
# En cas de kernel 'poly', le troisieme argument est le degre du polynome.
# Le quatrieme argument est le nombre maximal d'iterations (-1 pour infini)


### Import des modules
from sklearn import svm, datasets
import pylab as pl
import sys
import timeit

# Recuperation des arguments d'appel de ce script
LEARNING_RATE=float(sys.argv[1])
KERNEL=sys.argv[2]
DEGREE=int(sys.argv[3]) # si KERNEL = 'poly'
N_EPOCHS=int(sys.argv[4])

# Chargement du dataset Digits
digits = datasets.load_digits()
X_data = digits.images.reshape((digits.images.shape[0], -1))
y_data = digits.target

# Segmentation du dataset en trois jeux de donnees (entrainement, validation, test)
n_samples = len(X_data)
p60=int(0.6 * n_samples)
train_set_x = X_data[:p60] # 60 premiers % de la liste
train_set_y = y_data[:p60]
X_temp = X_data[p60:] # 40 derniers % de la liste
y_temp = y_data[p60:]
n_temp_samples = len(X_temp)
p50=int(0.5 * n_temp_samples)
valid_set_x = X_temp[:p50] # enrigistrements situes de 60% a 80% de la liste
valid_set_y = y_temp[:p50]
test_set_x = X_temp[p50:] # 20 derniers % de la liste
test_set_y = y_temp[p50:]

clf = svm.SVC(C=LEARNING_RATE, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=DEGREE, gamma='auto', kernel=KERNEL,
    max_iter=N_EPOCHS, probability=False, random_state=None, shrinking=True,
```

```
44      tol=0.001, verbose=False)

46  start_time = timeit.default_timer()
    clf.fit(train_set_x, train_set_y)
48  end_time = timeit.default_timer()
    print(('Score d\'entrainement: %f') % (clf.score(train_set_x,
        train_set_y)))
50
    clf.score(valid_set_x, valid_set_y)
52  print(('Score de validation: %f') % (clf.score(valid_set_x,
        valid_set_y)))

54  clf.score(test_set_x, test_set_y)
    print(('Score de test: %f') % (clf.score(test_set_x, test_set_y)))
56
    print('Taux d\'apprentissage (C);Kernel;Degree;Nombre d\'
        iterations;Score d\'entrainement;Score de validation;Score de
        test;Temps ecoule')
58  print(('Optimization complete;%f;%s;%d;%d;%.3f;%.3f;%.3f;%.1f') %
        (LEARNING_RATE,KERNEL,DEGREE,N_EPOCHS, clf.score(train_set_x,
        train_set_y),clf.score(valid_set_x, valid_set_y),clf.score(
        test_set_x, test_set_y),(end_time - start_time)))
```