Adaboost for Neural networks applied to an insurance Database

Lucien Ledune

16 april 2019

Abstract

This will be the abstract of the document

Contents

1 Introduction

Partout dans le monde, la facilité montante de l'accès à une puissance de calcul importante a motivé un grand nombre de changements dans la façon d'approcher les problèmes, et ce dans la plupart des domaines de recherche. Cette avancée technologique a permi d'utiliser des techniques de modélisations et de prédictions qui étaient jusqu'à lors trop longues à mettre en place pour avoir un réel intérêt pratique. Le secteur des assurances n'a pas été épargné par le phénomène et évolue avec son temps. Le machine learning[1], outil de modélisation très puissant, est maintenant facilement accessible et les compagnies d'assurances souhaient en tirer le meilleur parti en matière de prédiction et d'aide à la décision.
La modélisation de prédiction n'est pas nouvelle dans ce secteur, mais les algorithmes utilisés changent avec l'essort actuel de la technologie. Ainsi nous pouvons maintenant facilement utiliser des réseaux de neurones afin de répondre aux problèmes pour lesquels des algorithmes moins complexes étaient utilisés auparavant. Le but de ce travail est de montrer que ces nouveaux[2] algorithmes sont efficaces et représentent un espoir d'amélioration pour le futur. TODO : expliquer le mémoire rapidement.

2 Dataset

2.1 Présentation des données

Afin de réaliser ce travail, nous avons besoin d'une base de donnée adéquate, nous présentons celle-ci dans cette sous-section. La base de données est constituée d'informations sur les clients d'une compagnie d'assurance nommée Wasa, entre 1994 et 1998. Les véhicules assurées sont composés uniquement de motos. Il est difficile d'obtenir des données plus récentes à cause des clauses de confidentialité des assurances. La version originelle de ce jeu de données est utilisée dans une étude cas du livre "Non-life insurance pricing with GLM", écrit par Ohlsson et Johansson, et celle-ci est disponible sur le site web du livre[3]. Le jeu de données est constitué de 64505 observations des 9 variables suivantes :

[1] Apprentissage automatique : Méthodes statistique permettant à un ordinateur d'apprendre à résoudre un problème donné à l'aide d'une base de données pertinente.

[2] La plupart de ces algorithmes sont en fait assez peu récents mais étaient difficilement applicables en raison d'une puissance de calcul trop faible, citons par exemple le perceptron, élement de base des réseaux de neurones qui a été inventé dès 1957.

[3] http://staff.math.su.se/esbj/GLMbook/case.html

- OwnersAge : L'âge du conducteur.
- Gender : Le sexe du conducteur.
- Zone : Variable catégorielle représentant la zone dans laquelle le véhicule est conduit..
- Class : Variable catégorielle représentant la classe du véhicule. Les classes sont assignées dans une des 7 catégories selon le ratio : $EV = \frac{kW \times 100}{kg+75}$.
- VehiculeAge : L'âge du véhicule en années.
- BonusClass : Le bonus du conducteur, un nouveau conducteur commence à 1 et sera incrémenté à chaque année complète passée dans la compagnie sans sinistre déclaré, jusqu'à un maximum de 7.
- Duration : Le nombre d'année passées dans la compagnie.
- NumberClaims : Le nombre de sinistres.
- ClaimCost : Le coût des sinistres.

Table 1: Description de la variable "Zone".

Variable	Classe	Description
Zone géographique	1	Parties centrales et semi-centrales des trois plus grandes villes de Norvège.
	2	Banlieues et villes moyennes.
	3	Petites villes (à l'exception de celles des catégories 5 et 7).
	4	Villages (à l'exception de ceux des catégories 5 et 7).
	5	Villes du nord de la Suède.
	6	Campagnes du nord de la Suède.
	7	Gotland (Grande île).

2.2 Analyse exploratoire

Maintenant que les différentes variables ont été brièvement présentées et leur fonction plus claire, nous allons maintenant passer à l'analyse exploratoire de celles-ci. Le but de cette analyse est de mieux comprendre les données qui serviront à entraîner les différents algorithmes, ainsi que de repérer d'éventuelles anomalies. Durant l'analyse exploratoire d'une base de données, il est important de regarder la distribution des variables, celle-ci nous donne beaucoup d'informations quant aux données.

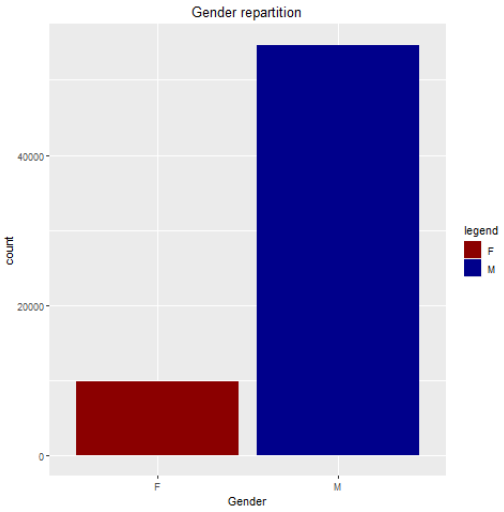


Figure 1: Distribution Gender

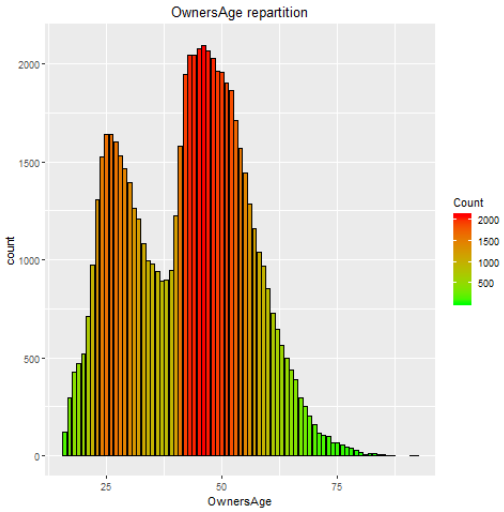


Figure 2: Distribution OwnersAge

Sur ces deux premières figures nous observons respectivement les distributions des variables Gender et OwnersAge. La première chose que nous pouvons voir est la grande disparité entre le nombre d'hommes de et femmes clients de l'assurance. Notre jeu de données est composé d'hommes pour la grande majorité. Pour ce qui est de la variable OwnersAge, nous constatons que les valeurs sont réparties entre 16 et 92 ans, avec deux "pics" vers 25 et 45 ans. Les valeurs maximales et minimales de nos variables continues seront importante pour la suite, car elles sont nécessaires afin d'appliquer une normalisation des données, qui sera discutée plus tard dans ce travail. Ci dessous les distributions des variables VehiculeAge et Zone :

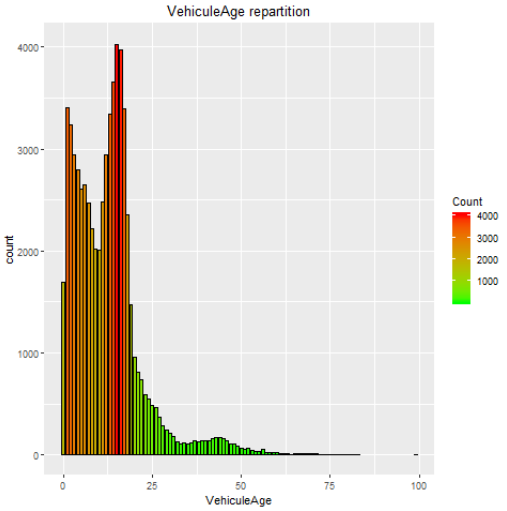


Figure 3: Distribution VehiculeAge

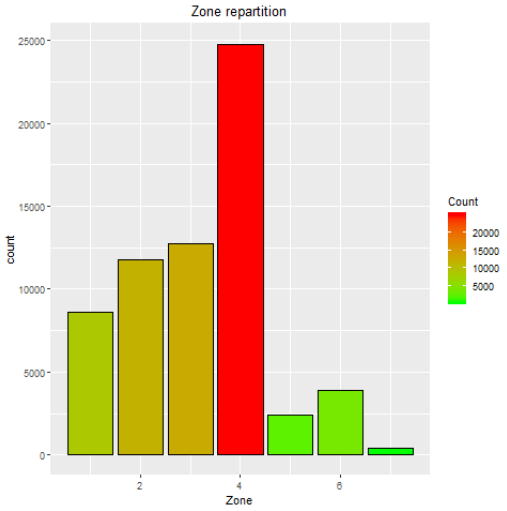


Figure 4: Distribution Zone

La distribution de vehicule est indiquée en années, et nous pouvons donc observer que si la plupart des véhicules assurés ont moins de 20 ans, un grand nombre de ceux-ci sont bien plus vieux, avec comme maximum 99 ans. Il est possible que ce véhicule soit considéré comme outlier et il sera reconsidéré dans la partie preprocessing. La distribution de la variable Zone est intéressante, elle nous révèle que la plupart des véhicules assurés sont conduits dans des villages, mais aussi que très peu d'entre eux le sont dans le Gotland. Ceci n'est pas surprenant puisque la population de la Suède est d'environ 10 millions d'habitants, pour seulement 60.000 habitants la région du Gotland. Les classes 5 et 6 sont elles aussi minoritaires, cela était aussi à prévoir puisque ces catégories représentent le nord de la Suède alors que la plupart de la population vis dans le sud du pays.

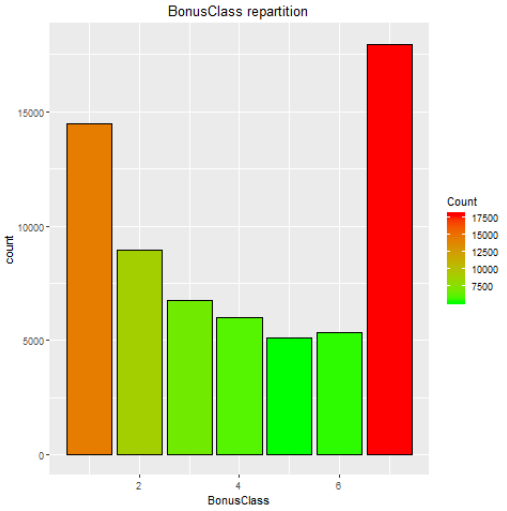


Figure 5: Distribution BonusClass

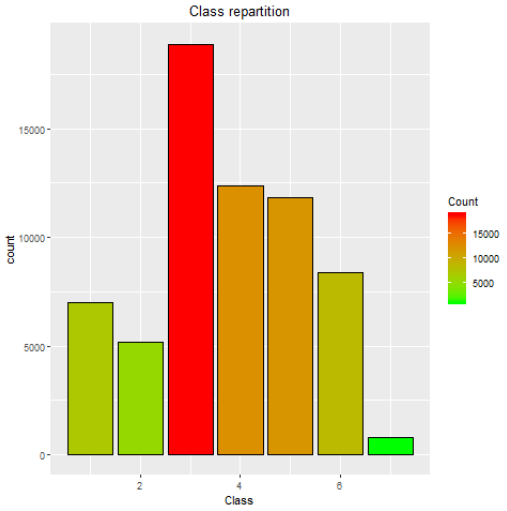


Figure 6: Distribution Class

Les classes de bonus les plus fréquentes sont les classes 1 et 7, respectivement le minimum (entrée dans la compagnie d'assurance) et le maximum (client fidèle depuis sept années au minimum).
Pour la variable Class nous observons qu'assez peu de véhicules appartiennent à la catégorie la plus puissante. En fait, la plupart des véhicules se situent dans les classes 3, 4 et 5, ce qui montre que les vehicules les moins puissants et les plus puissants sont minoritaires. Les graphiques suivants nous montrent la répartitions des sinistres par bonus et par classe de véhicule.

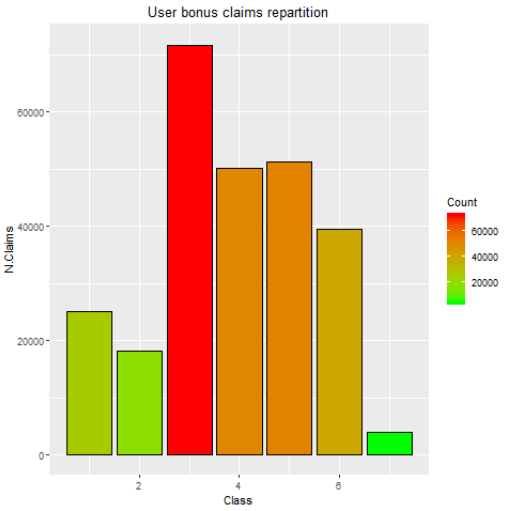


Figure 7: Bonus Claims

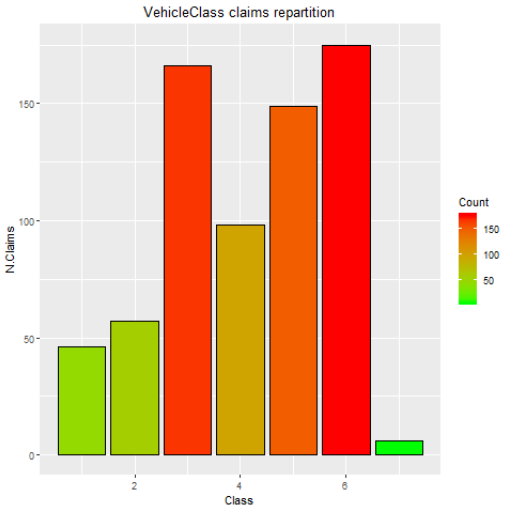


Figure 8: Class Claims

Il parait logique de supposer que plus un client a un bonus élevé, moins celui-ci causera d'accident, puisque le bonus monte uniquement si le client parvient à compléter une année sans causer d'accident. Cependant en observant la figure 7, il apparait que la majorité des cas d'accidents sont déclarés par des clients appartenant aux classes 3 à 6 de bonus. Ce qui est d'autant plus étonnant lorsque l'on associe ce résultat avec la distribution de la variable Class (figure 5) : les classes 3 à 6 sont celles contenant le moins d'utilisateurs. Les clients appartenant à la classe de bonus 7 semblent cependant causer très peu d'accidents malgré le fait qu'ils soient la classe de bonus majoritaire.
Les résultats de la figure 8 sont moins surprenant : plus un véhicule est puissant, plus le risque d'accident sera important. Les déviations de cette régle par les classes 3 et 7 sont expliquées par la distribution de la population à travers les différentes classes (figure 6), ainsi il y a peu d'accidents pour les véhicules de classe 7 simplement car ceux-ci sont peu nombreux, une conclusion similaire peut être énoncée pour la classe 3.

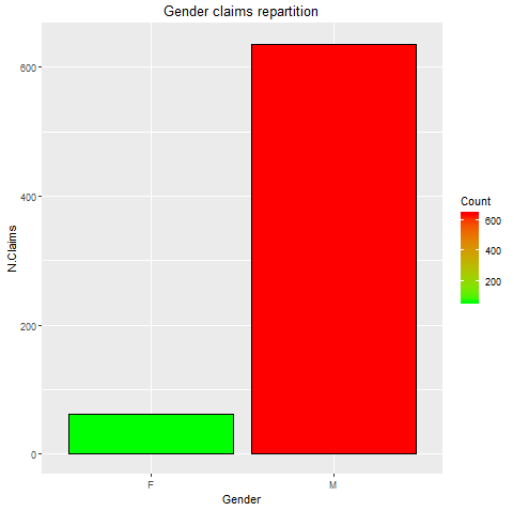


Figure 9: Gender claims

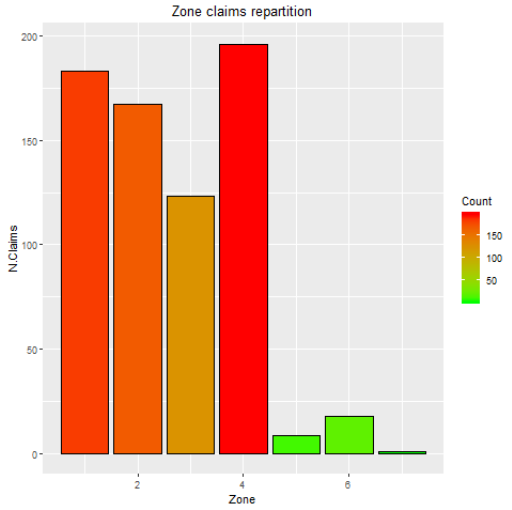


Figure 10: Zone claims

La répartition des sinistres par sexe indique que les hommes sont plus susceptibles de causer des accidents que les femmes. Il faut prendre en compte que les hommes sont bien plus nombreux que les femmes dans nos données, mais la conclusion ne change pas puisque la proportion de femmes est de 18%, alors que celles-ci ne causent que 8.7% des sinistres. Sur la figure 10, nous observons que les classes 1 et 3 sont celles qui déclarent le plus de sinistres. La première classe étant plutôt minoritaire (figure 4) nous observons que les personnes habitant dans les parties centrales et semi-centrales de Norvège

semblent causer bien plus d'accidents que les autres catégories, la même conclusion peut être faite pour la deuxième classe (Banlieues et villes moyennes). Pour ce qui est de nord de la Suède, nous constatons l'inverse puisque ceux-ci semblent causer assez peu d'accidents.

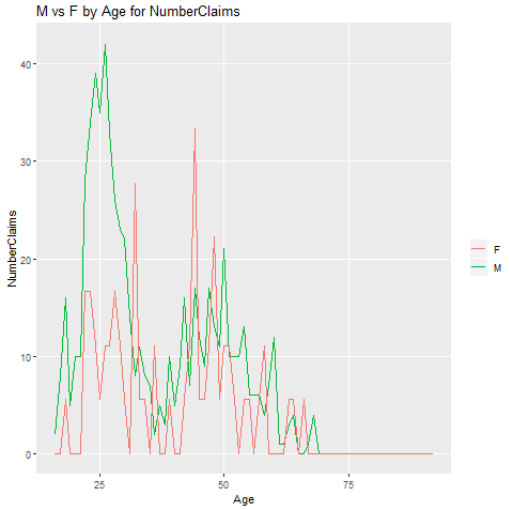


Figure 11: Gender claims by Age

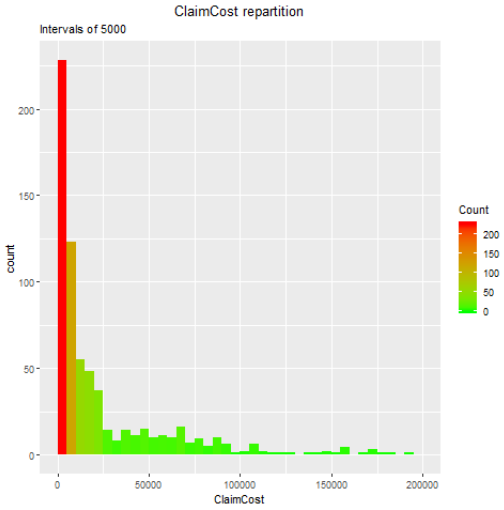


Figure 12: Distribution ClaimCost

La figure 11 a pour but de mieux comprendre cette disparité entre les hommes et les femmes dans la déclaration des sinistres. Une comparaison par âge permet de mettre en évidence une information importante. En effet, si les hommes semblent bien causer plus d'accidents que les femmes jusqu'à la treintaine, cette tendance s'égalise passé ce cap. Enfin, la figure 12 nous informe de la distribution de la variable ClaimCost, représentant le coût total des sinistres d'un client. Il est important de noter qu'il s'agit bien de la somme de tous les sinistres déclarés du client, ainsi si une personne a reçu une compensation de l'assurance pour plusieurs sinistres, la variable contient la somme du coût de tous ces sinistres déclarés. Il semble que la grande majorité des clients ne dépassent pas 5000€ de compensation venant de l'assurance (Le graphique ne montre que les valeurs non-nulles, ainsi les clients n'ayant jamais déclaté de sinistres ne sont pas comptés dans l'intervalle $[0, 5000]$. La distribution de cette variable montre bien qu'assez peu de clients dépassent les 20.000€ de compensation, mais certains peuvent monter à près de 200.000€.

2.3 Assurances

Dans cette sous-section seront expliquées les particularités de l'analyse de données dans le cadre de l'assurance, et plus précisement dans le cas qui nous intéresse ici.

2.3.1 Principe

Un contrat entre l'assureur et l'assuré implique le payement d'une prime d'assurance par le second, en échange de compensations lorsqu'un sinistre est déclaré. Le fait de regrouper un grand nombre de clients va avoir un effet stabilisateur de variance pour l'assureur. En effet il est difficile de prédire précisemment le nombre d'accidents qu'un assuré aura, cependant plus le nombre d'assurés sera élevé et plus le nombre (total) de sinistres sera prévisible. Ceci est du à la loi des grands nombres qui implique :

$$\bar{X} \to \mu \, when \, n \to \infty$$

La moyenne de l'échantillon $\bar{X}$converge vers le moyenne de la population μ lorsque la taille de l'échantillon augmente.
Dans le cadre de l'assurance, les variables explicatives sont aussi appelées facteur d'évaluations, les deux termes seront utilisés indifférement au cours de ce travail.

2.3.2 Ratio

Il est important de comprendre ce que l'on cherche à expliquer avec un modèle, aussi il faut savoir que cette analyse se base sur un ratio. Un ratio Y est un rapport entre une réponse X et une exposition[4] v.

$$Y = X/v$$

Il existe différents ratios mais ici ne sera présenté que celui utilisé dans l'analyse à venir : La fréquence des réclamations. Il s'agit du rapport entre le nombre de réclamations et la durée du contrat, nous obtenons ainsi une fréquence qui sera la variable prédite par nos modèles.

[4] L'exposition peut être le nombre de sinistres ou la durée du contrat.

2.3.3 Distribution du ratio

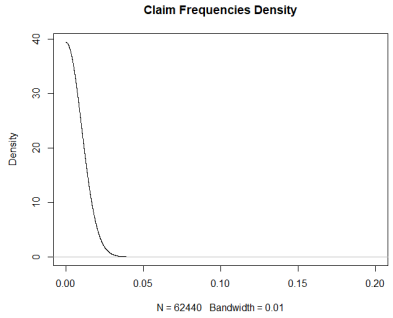
Pour la création d'un modèle cherchant à prédire la fréquence des réclamations c'est souvent la distribution Poisson qui est utilisée. Une loi poisson de paramètre λ implique :

$$Y \sim Poisson(\lambda)$$

$$E[Y] = \lambda$$

$$Var[Y] = \lambda$$

Afin de vérifier que la distribution de la fréquence des sinistres suit bien une loi de poisson celle-ci est affichée. En effet la plupart des valeurs se situent près de 0, les valeurs supérieures à 1 sont présentes mais extrêmement rares et ne sont donc pas affichées afin d'observer plus précisemment les valeurs types.



La loi de poisson fait partie de la famille des modèles à dispersion exponentielle (ED), une catégorie regroupant un grand nombre de lois statistiques connues comme la loi normale, gamma ou encore binomiale. Une variable aléatoire Y suit une dispersion exponentielle si sa fonction de distribution suit la forme suivante :

$$f_{Y_i}(y; \theta_i; \phi) = exp\left\{\frac{y\theta_i - a(\theta_i)}{\phi/v_i}\right\} c(y, \phi, v_i)$$

- θ_i: paramètre dépendant de i.
- ϕ: paramètre identique pour tous les i. C'est le paramètre de dispersion.
- $c(.)$ est indépendante du paramètre θ_i.

2.4 Preprocessing

Le preprocessing des données est une étape très importante lorsque nous travaillons avec ce type de données et celui-ci n'est pas à négliger. Il consiste a préparer les données pour les algorithmes, afin que ceux-ci puissent fonctionner de manière optimale. Dans ce travail, plusieurs méthodes de preprocessing ont été utilisées.

2.4.1 Variables binaires (Dummy variables)

Certaines de nos variables sont catégorielles, et plus particulièrement sont des variables non-ordinales. Cela signifie que les différentes catégories ne peuvent pas être ordonnées, il s'agit purement de l'assignation d'un client à un groupe donné. Ce type de variable ne peut pas être encodé tel quel et doit être transformé en une série de variables binaires. La conversion d'une variable catégorielle en variable binaire consiste à créer $n-1$ nouvelles variables dites "binaires" et qui prendront pour valeur 1 si le client appartient à cette catégorie, 0 sinon. Par exemple si x_{ij} représente la variable associée au client i et à la variable j, nous pouvons écrire :

$$x_{ij} = \begin{cases} 0 \text{ if } x_i \notin j \\ 1 \text{ if } x_i \in j \end{cases}$$

Les variables Gender, Zone, Class, et BonusClass peuvent être transformées de la sorte. Ce travail sera effectué avec le package caret[5] de R. La fonction incluse dans ce package crée n variables binaires, la dernière sera donc supprimée car celle-ci ne représente aucune information. Prenons pour exemple la variable Gender. Si $x_{iGender} = 1$ alors le client est une femme. Cependant nous ne devons pas créer une deuxième variable binaire puisque si $x_{iGender} = 0$ nous pouvons en déduire que le client est un homme. Ceci explique pourquoi nous n'avons besoin que de $n-1$ variables binaires pour représenter toute l'information de n catégories appartenant à une variable catégorielle.

2.4.2 Normalisation

Les données quantitatives ne doivent pas être transformées en variables binaires (bien que celles-ci peuvent être converties en intervalles puis en variables binaires, ce ne sera pas le cas dans le cadre de ce travail). Cependant cela ne signifie pas qu'aucune méthode de preprocessing ne doit être appliquée

[5] Classification And REgression Tools : Package de machine learning pour R.

à ces données. La normalisation des données quantitatives aide certains algoithmes à converger plus rapidement et peut même parfois améliorer leur précision. Cette normalisation des données est particulièrement importante pour les réseaux de neurones, car l'optimisation de ceux-ci est basée sur les résultats des fonctions d'activation[6] utilisées. Celles-ci varient rapidement entre 0 et 1 (ou -1 et 1) et sans la normalisation des données leurs réponses aux valeurs extrêmes seraient localisées dans les extrémités, ce qui peut empêcher l'algorithme de converger dans les cas les plus graves.
Plusieurs méthodes peuvent être utilisées afin de procéder à la normalisation des données. Cependant nous ne décrirons que celle qui sera utilisée dans le cadre de ce travail : la normalisation Min Max.
Elle consiste (pour une variable j) à convertir l'ensemble des données sur l'intervalle [0,1] en se servant de $Max(x_j)$[7] et $Min(x_j)$[8] par la formule :

$$z_{ij} = \frac{x_{ij} - min(x_j)}{max(x_j) - min(x_j)}$$

Cette transformation sera appliquée aux variables OwnersAge et VehiculeAge.

2.4.3 Outliers

Lors de l'encodage des données, il arrive que certaines valeurs des variables quantitatives soient assez extrêmes et semblent fort distantes des autres observations. Dans ce cas nous pouvons soit garder la valeur si celle-ci semble ajouter de l'information au modèle ou la supprimer dans le but d'une meilleure généralisation par celui-ci. Dans la section 2.2 une donnée a particulièrement attiré notre attention comme outlier potentiel : une moto de 99 ans. Pour celle-ci nous pouvons entrevoir deux possibilités :

- Il peut simplement s'agir d'une erreur d'encodage, auquel cas l'observation doit être supprimée pour éviter de biaiser le modèle.
- Il peut également s'agir d'une vraie moto de type "ancêtre", cependant ce genre de véhicule ne se conduit pas de la même manière qu'un véhicule utilisé au quotidien, et on peut supporser que les risques d'accidents sont bien moindres. Dans ce cas-ci la suppression de la donnée n'est pas obligatoire mais peut aider à une meilleure généralisation du modèle.

[6] Voir chapitre 3
[7] Valeur maximale prise par la variable j.
[8] Valeur minimale prise par la variable j.

Puisqu'aucun moyen simple de vérifier cette hypothèse n'est disponible, celle-ci sera supprimée de la base de donnée car la perte d'information est moindre. La suppression de trop d'information peut cependant nuire au modèle, les données extrêmes mais non-aberantes seront donc conservées.

2.4.4 Durée de contrat

Pour le bon fonctionement du modèle, certaines autres données ont du être modifiées car celles-ci entrainaient des erreurs dans la fonction de déviance utilisée pour la calibration des poids des réseaux de neurones. Les observations incriminées sont celles ayant une durée de contrat nulle. Ces valeurs nulles de durée de contrat sont peu cohérentes (impossible d'avoir un sinistre déclaré au cours du contrat si celui-ci n'est pas encore en vigueur) et elles seront donc changées par la valeur correspondant à un jour ($1/365 \simeq 0.00274$). Certaines de ces observations n'ont cependant pas une valeur nulle pour la variable NbClaims, ce qui signifie que des sinistres auraient été déclarés dans des contrats dont la durée est nulle. Ce cas de figure n'étant pas possible, elles seront supprimées car il s'agit probablement d'erreur d'encodage. De plus changer la valeur de la durée du contrat par 0.00274 dans ce cas reviendrait à considérer une fréquence de sinistre de $\simeq 365$ soit un sinistre par jour ce qui pourrait grandement biaiser le modèle.

3 Algorithmes

Dans cette section, les différents algorithmes utilisés lors des analyses seront présentés à des fins de compréhension du travail. .....
Afin d'effectuer un travail de prédiction en matière de régression ou de classification, il est nécessaire d'avoir un outil puissant à disposition : l'apprentissage automatique ou "machine learning". Il s'agit d'un ensemble de méthodes basées sur des outils statistiques permettant la création de modèles pouvant être utilisés pour la prédiction. (réf wiki)

3.1 Explication d'un modèle

3.1.1 Définitions et apprentissage

Avant de pouvoir rentrer dans les détails, il est important de définir les notations qui seront utilisées, de définir ce qu'est un modèle et comment celui-ci se construit. La construction d'un modèle est spécifique pour la résolution d'une tâche T, que l'on cherche à améliorer par rapport à une métrique P

grâce à des expériences[9] Q. La base de données est composée de X les variables explicatives et de Y la variable expliquée. Dans notre cas :

- T : Déterminer la fréquence des sinistres.
- P : Déviance (Explications dans la section 3.1.2).
- $Q = \{(x_1, y_1), ..., (x_n, y_n)\}$: Base de données $X \to Y$

Un modèle cherche à approximer une fonction cible (choisie), dans le cas étudié $f : X \to \mathbb{R}^+$
Dans notre étude seuls les modèles supervisés[10] seront abordés. Deux éléments sont nécessaires pour la création d'un modèle, une base de données et un algorithme supervisé (choix du modèle). La base de données est composée de N entrées des variables explicatives et de la variable expliquée y_i où i représente l'individu et où x_{ij} représente la valeur prise pour la variable j pour l'individu i. Le modèle va se servir des valeurs prises par les x_{ij} pour tenter d'expliquer y_i. Pour ce faire l'algorithme doit subir une phase d'apprentissage, aucours de laquelle celui-ci va recevoir des entrées provenant de la base de données et se servir de celles-ci pour "apprendre" et se calibrer sur le problème que l'on cherche à résoudre. Cette phase d'apprentissage est différente d'un algorithme à un autre. Une fois cette phase d'apprentissage terminée le modèle ainsi créé peut être utilisé pour la prédiction (après que celui-ci ait été validé, voir sous-sections suivantes). Les valeurs prédites par l'algorithmes seront notées $\hat{y}_i$.

3.1.2 Évaluation et fonctions de perte

Pour l'apprentissage, les algorithmes supervisés se basent sur un critère mesurant l'erreur d'un modèle : la fonction de perte. Ce critère permet la comparaison entre différents modèles et un choix optimal parmis ceux-ci, en sélectionnant celui qui minimise l'erreur commise dans les prédictions. Il existe un grand nombre de critères pouvant servir pour l'évaluation des modèles mais le choix de celui-ci doit se faire en fonction du problème adressé et de la base de données. Le choix de ce critère est important car il peut impacter l'efficacité des modèles créés.
Le critère peut être défini comme $f(y, \hat{y})$ une fonction dépendant des valeurs prises par y et $\hat{y}$ mesurant la différence entre les prédictions et la réalité d'une certaine manière. (https://en.wikipedia.org/wiki/Loss_function) Une

[9] Données.
[10] Algorithmes nécessitant une base de données $X \to Y$ pour l'apprentissage.

fonction de perte intuitive est donnée en guise d'example par la racine de l'erreur quadratique moyenne (RMSE).

$$RMSE(\theta, \hat{\theta}) = \sqrt{MSE(\theta, \hat{\theta})} = \sqrt{E((\hat{\theta} - \theta)^2)}$$

Le RMSE est sûrement un des critères les plus utilisés aujourd'hui (y compris pour les réseaux de neurones), cependant utiliser celui-ci reviendrait à considérer le ratio de la fréquence des réclamations comme étant distribué selon une loi gaussienne. Les résultats s'en trouvent alors biaisés car les propriétés statistiques de la variable expliquée ne sont pas prises en compte. Afin de palier à ce problème, un autre critère statistique sera utilisé : la déviance de Poisson. La déviance est utilisée la plupart du temps pour des tests d'hypothèses et elle joue un role important dans l'analyse des distributions de la famille exponentielle. Elle peut être vue comme une forme de distance (d'où son utilisation en temps que critère). Aussi : (wiki deviance)

- D^*, la déviance mise à l'échelle, un test entre les maximums de vraisemblance du modèle entraîné et du modèle saturé[11].
- $l(\hat{y}_i)$, le log-vraisemblance de $\hat{y}_i$.
- $l(y_i)$, le log-vraisemblance de y_i.

$$D^*(y_i, \hat{y_i}) = 2\left(l(y_i) - l(\hat{y_i})\right)$$

En utilisant la fonction de distribution des ED vue précédemment, le résultat suivant est obtenu :

- $a'(\theta) = e^\theta$
- $h(y) = ln(y)$

$$D^*(y_i, \hat{y_i}) = \frac{2}{\phi} v_i(y_i h(y_i)) - a(h(y_i)) - y_i h(\hat{y}_i) + a(h(\hat{y}_i)))$$

La déviance non mise à l'échelle peut ensuite être obtenue en multipliant cette formule par le paramètre de dispersion ϕ (voir section 2.3.3). La dérivation de cette déviance non mise à l'échelle appliquée à la loi de poisson résulte

[11] Modèle où les $\hat{y}_i$sont définis comme les véritables valeurs y_i,le maximum de vraisemblance de ce modèle est le meilleur que nous pouvons obtenir du fait que les $y_i$sont parfaitement prédits.

en la déviance de Poisson non-mise à l'échelle, c'est ce critère qui sera utilisé pour l'apprentissage de nos modèles.

$$D(y_i, \hat{y}_i) = 2v_i(y_i ln(y_i) - y_i ln(\hat{y}_i) - y_i + \hat{y}_i), \ \ \forall y_i > 0$$

La base de donnée utilisée portant sur des événements rares, beaucoup de cas $y_i = 0$ seront étudiés, en modifiant la formule ci dessus la formule adéquate pour ce cas particulier est donnée par :

$$D(y_i, \hat{y}_i) = 2v_i\hat{y_i}, \ \ \forall y_i = 0$$

3.1.3 Overfit, AIC et BIC

Il est important de comprendre le but derrière la création d'un tel modèle. Nous cherchons à approximer une fonction de réponse Y à l'aide de ce modèle. Afin que cette fonction sois au mieux approchée, il est important que le modèle reste général. En effet le but final du modèle est de prédire des valeurs de Y pour des nouvelles entrées X_i, ce qui amène à des méthodes permettant de vérifier que cette fonction est correctement approximée. Plus particulièrement nous chercons à éviter un maximum l'overfit. L'overfit survient lorsqu'un modèle approxime bien les valeurs du jeu de données utilisé pour l'entraînement mais que celui-ci est incapable de déterminer des valeurs cohérentes pour des nouvelles entrées, il peut être notamment causé par un trop grand nombre de variables explicatives ou une phase d'apprentissage mal configurée. Le modèle est alors inutilisable dans un scénario réél.
Afin de choisir des modèles limitant l'overfit, deux nouveaux critères seront introduits : AIC et BIC. Comme les autres critères vus, ce sont des mesures de la qualité d'un modèle statistique. Il est (presque) toujours possible d'améliorer les résultats d'un modèle sur les critères basiques en augmentant le nombre de variables explicatives. Cependant cette pratique augmente les risque d'overfit, c'est la raison pour laquelle dans la création d'un modèle la parcimonie[12] du modèle est importante. Ces deux critères permettent de prendre en compte le nombre de variables explicatives dans le calcul de la qualité de modèle.

- $AIC = 2k - 2ln(L)$ [13]
  (Où k est le nombre de paramètres à estimer du modèle et L est le maximum de la fonction de vraisemblance du modèle (edit copy))

[12] Principe consistant à utiliser le moins de variables explicatives possible pour expliquer un phénomène afin de réduire le phénomène d'overfit.
[13] Akaike Information Criterion

- $BIC = k\,ln(N) - 2ln(L)$ [14]
  (Où N est le nombre d'observations dans l'échantillon)

3.1.4 Validation croisée (K-fold)

La valeur de certains paramètres d'un modèle pouvant être générée dans un premier temps de manière aléatoire lors de l'initialisation avant d'être optimisés, et la forme du jeu de données pouvant aussi influencer les résultats, deux modèles "identiques" n'auront pas toujours le même résultats pour les critères statistiques utilisés. Il est aussi parfois difficile de détecter l'overfit, c'est la raison pour laquelle la validation croisée est utilisée. Généralement la base de données utilisée est divisée en un jeu d'entraînement et un jeu de test qui ne sera pas utilisé pour l'entraînement mais pour calculer les résultats du modèle aux différents critères statistiques. Dans le cas étudié ici, les événements sont rares, c'est pourquoi il est souvent préférable de travailler avec la base de donnée complète en évitant cette division. C'est pour palier à ce problème que la validation croisée sera utilisée.
Le principe de base est de diviser le jeu de données en K ensembles de taille égale. Un modèle sera ensuite entrainé pour chaque ensemble en utilisant cet ensemble comme jeu de test et l'ensemble des autres ensembles comme jeu d'entraînement. Le résultat de la métrique est alors la moyenne des résultats des différents ensembles. Plus formellement :

- $Q = \{(x_1, y_1), ..., (x_n, y_n)\}$, la base de données.
- K, le nombre d'ensembles créés.
- M, l'algorithme d'apprentissage.
- $E(M, Q)$, la fonction de perte dépendant de y et $\hat{y}$, calculée sur Q avec le modèle M.

[14] Bayesian Information Criterion

Algorithm 1 Validation croisée (K-folds)

begin
Diviser Q en K sous-ensembles $Q_1, ..., Q_k$
for $k = 1, ..., K$
test $= Q_k$
train $= Q \setminus Q_k$
$M_k = M(train)$//Entraînement du modèle.
$E_k = E(M_k, test)$ //Erreur du modèle M_k sur Q_k.
endfor
$E(Q) = K^{-1} \sum_{i=1}^{i=k} E_k$//Moyenne des erreurs
return $E(Q)$
end

Cette méthode permet de limiter l'overfit et de le détecter dans la plupart des cas. La valeur de la métrique donnée par la validation croisée dispose d'une variance inférieure aux résultats donnés sans celle-ci. Comme dit précédemment la base de donnée est constituée d'événements rares et cette méthode sera donc préférée à la traditionnelle décomposition entrainement/test.

3.2 GLM : modèle linéaire généralisé

3.3 Réseaux de neurones

Avant de comprendre comment cet algorithme fonctionne d'un point de vue mathématique, il est intéressant de comprendre de quoi est inspiré son fonctionnement. Comme son nom peut l'indiquer, le réseau de neurones artificiels est un algorithme d'apprentissage automatique inspiré du fonctionnement du cerveau humain. Un neurone biologique est composé du corps de la cellule qui contient les composants les plus importants, de plusieurs extrémités nommées les dendrites, ainsi que de l'axone (une extension du neurone pouvant être des milliers de fois plus longue que le corps du neurone). L'axone se subdivise en plusieurs branches appelées synapses. Les neurones fonctionnent en recevant des impulsions électriques appelées "signaux", et lorsqu'un neurone reçoit un certain nombre de signaux en même temps, celui-ci va déclencher son propre signal.

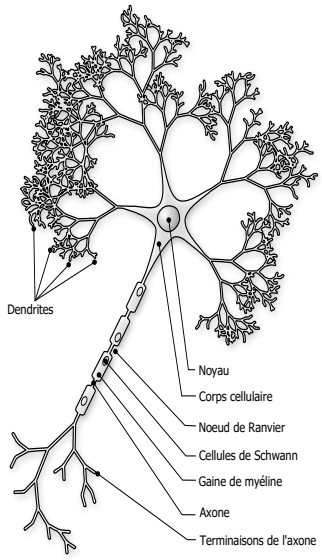


Figure 13: Schéma d'un neurone biologique.

3.3.1 Perceptron

Le perceptron est le modèle à la base des réseaux de neurones artificiels modernes, celui-ci a été inventé dès 1957 par Frank Rosenblatt. Il s'agissait alors d'un algorithme permettant l'apprentissage automatique par l'erreur (apprentissage supervisé). Son fonctionnement est très similaire à celui d'un neurone humain, puisqu'il en est inspiré. Mathématiquement, un perceptron est la somme d'un biais et de multiplications entre les valeurs des variables et les poids du perceptron passant par une fonction dite "d'activation" qui détermine la réponse du neurone. La fonction d'activation représente le potentiel d'activation du neurone. L'exemple le plus simple serait une fonction renvoyant 1 si l'entrée est positive et 0 (ou -1) si l'entrée est négative (bien que cette fonction ne sera pas utilisée dans nos algorithmes car non différentiable, ce qui est un problème pour la calibration, voir sections suivantes), c'est d'ailleurs cette fonction qui a été utilisée en premier pour les perceptrons.

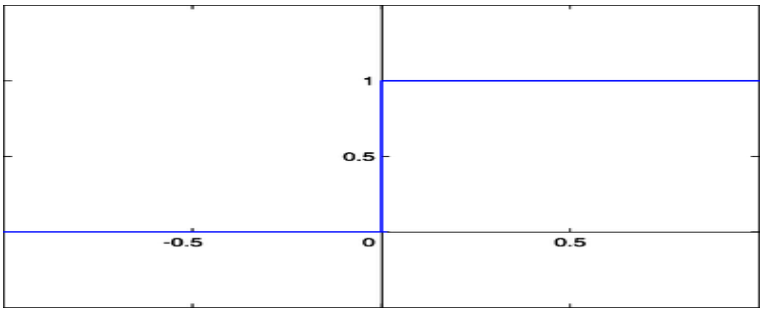


Figure 14: Exemple de fonction d'activation : Step function

Pour en revenir au perceptron, notons que les données d'un individu sont considérées comme un vecteur X_i et que l'ensemble des données constituent la matrice de données X. L'attribut j de la personne i est noté x_{ij}. Le vecteur des poids du perceptron est noté W et w_n représente le poids n. Il existe aussi un biais noté w_0 qui n'est pas multiplié par une valeur $x_i$mais par 1, il n'intéragit donc pas avec les valeurs prises par les variables. La fonction d'activation est quant à elle notée $\varphi(.)$. Il est important de noter que le nombre de poids correspond à J, soit le nombre de variable explicatives utilisées. Un perceptron peut être représenté comme ceci :

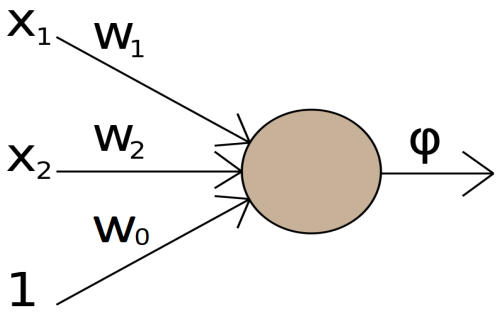


Figure 15: Représentation d'un perceptron

La manière dont un perceptron calcule la réponse Y peut être décomposée en deux étapes :

- Les poids sont multipliés par les valeurs des entrées à laquelle on ajoute le biais : $\omega = \sum_{j=1}^{J} x_j w_j + w_0$
- La valeur ainsi obtenue est prise comme argument dans la fonction d'activation : $Y = \varphi(\omega)$

C'est donc cette réponse Y qui sera considérée comme le résultat du perceptron. Le perceptron reste cependant un modèle simple et il ne peut être utilisé que pour des problèmes relevant de classification séparable linéairement, et son fonctionement pourrait être schématisé par la recherche d'un seuil représentant la valeur des variables à partir de laquelle un exemple serait considéré comme appartenant à une classe ou à l'autre.
Lorsqu'il est question d'apprentissage automatique, cela signifie en fait que les paramètres du modèle sont optimisés en fonction de notre base de données. Pour le perceptron cela signifie trouver les valeurs optimales w pour un problème spécifique. Pour être plus précis, l'algorithme va prendre en entrée une ligne de notre base de données et faire une prédiction. Si cette prédiction est correcte alors les poids ne sont pas modifiés, cependant dans le cas ou la prédiction est incorrecte nous utilisons une règle de mise à jour des poids, afin de donner plus d'importance aux entrées qui auraient rendu la prédiction correcte. Ce procédé sera expliqué plus en détail dans la section suivante, mais nous pouvons déjà observer l'équation de mise à jour des poids d'un perceptron simple :

$$w_i^{next} = w_i + \eta(\hat{y_j} - y_j)x_i$$

- w_i le poids du neurone i
- η le taux d'apprentissage, il s'agit d'un paramètre déterminant à quel point nous devons modifier les poids lors d'une erreur, celui-ci est inclus dans l'intervalle $[0, 1]$.
- $\hat{y_j}$ la prédiction du perceptron
- y_j la valeur cible

Comme mentionné précédemment, le perceptron reste cependant un modèle simple, et celui-ci est incapable de résoudre certains problèmes pourtant basiques. L'exemple le plus connu est le problème XOR[15], insolvable par le perceptron et mentionné par Minsky et Papert dans leur papier intitulé "Perceptrons" qui met en avant un certaines faiblesses du perceptron. Suite à ces découvertes la recherche dans le domaine va être mise de côté. Cependant il est plus tard apparu que ces faiblesses pouvaient pour la plupart (y compris le problème XOR) être éliminées en empilant plusieurs perceptrons en couches de neurones artificiels. (livre scikit)

[15]"OU" exclusif, opérateur logique de l'algèbre de Boole. Pour deux opérandes, celui-ci permet à chacun d'avoir la valeur VRAI, mais pas les deux en même temps.

3.3.2 Perceptron multicouche

3.3.2.1 Fonctionnement

Le perceptron multicouche, plus communément appelé MLP[16] est une structure composée de couches, elles-mêmes formées par plusieurs perceptrons. Pendant longtemps, les chercheurs ont tenté de trouver une manière efficace d'effectuer l'entraînement de ce type de réseau sans succès, jusqu'au jour où D. E. Rumelhart a découvert une méthode appelée la rétropropagation (1986) qui sera expliquée plus loin dans ce chapitre. Le schéma ci-dessous montre la structure d'un tél réseau :

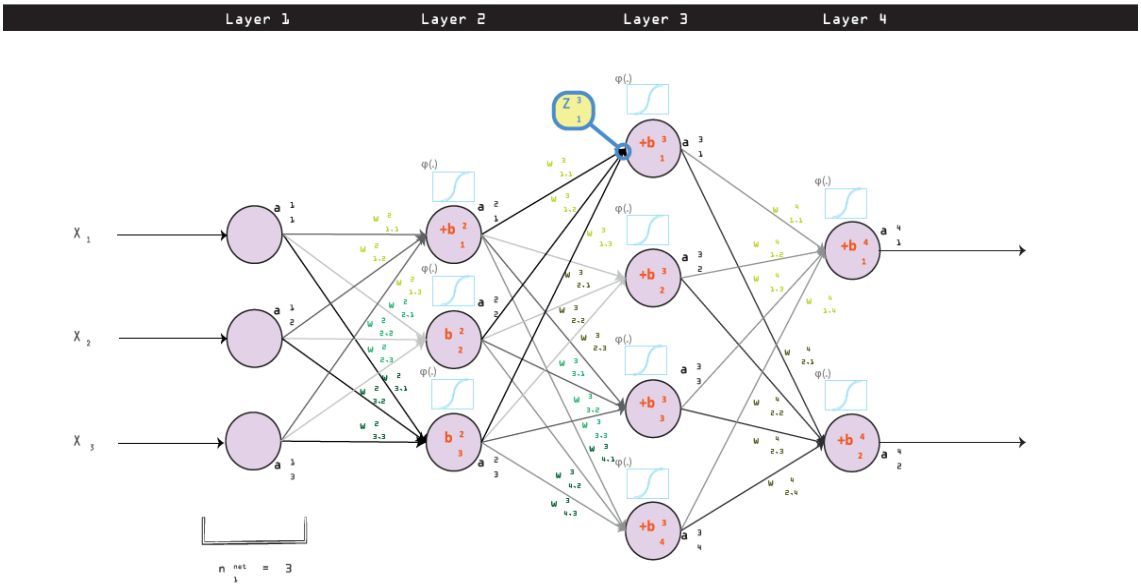


Figure 16: Représentation d'un MLP "feed-forward".

Un réseau de ce type résout la plupart des problèmes du perceptron lorsqu'il est utilisé seul et compose un modèle beaucoup plus puissant, capable de détecter les relations non-linéaires et de résoudre un grand nombre de problèmes de classification et de régression. Ce type de réseau est utilisé dans un grand nombre d'applications récentes telles que les voitures autonomes, la reconnaissance vocale ou encore la vision artificielle[17]. Comme observable sur le graphique, l'information circule dans un seul sens sur notre schéma.

[16] Multilayer perceptron
[17] Computer vision.

Il faut savoir qu'il existe différent types de MLP, dont certains avec des relations circulaires entre les neurones. Cependant ce type de réseau ne sera pas utilisé dans ce travail et ne seront donc pas vus en détails, les réseaux de neurones artificiels utilisés seront uniquement des réseaux feed-forward ; l'information arrive à l'entrée du réseau, transite à travers celui-ci dans une direction unique, et le modèle nous renvoie une réponse $\hat{y}_i$ en sortie.
Un réseau tel que celui-ci est composé de plusieurs couches à travers l'information passe successivement. La première couche est la couche d'entrée, celle-qui va recevoir les données. Ensuite, il y a une ou plusieurs couches cachées[18] et enfin le couche de sortie renvoyant la réponse du réseau. Le nombre de neurones par couche et le nombre de couches cachées sont tout deux des hyperparamètres[19] à définir.
Le fonctionement de ce réseau est similaire à celui du perceptron, cependant son architecture est plus complexe et nécessite certains changements dans les notations utilisées, aussi https://stats.stackexchange.com/questions/154879/a-list-of-cost-functions-used-in-neural-networks-alongside-applications :

- w^i_{jk} : poids du neurone k dans le couche $(i-1)$ vers le neurone j de la couche i.
- b^i_j : le biais du neurone j dans la couche i.
- a^i_j : la valeur de la fonction d'activation du neurone j dans la couche i.
- $z^i_j = \sum_k (w^i_{jk} \cdot a^{i-1}_k) + b^i_j$: La valeur d'activation avant de passer par la fonction d'activation.
- $\varphi(.)$: la fonction d'activation.
- n^{net} : le nombre de couches du réseau (entrée et sortie comprises).
- n^{net}_j : le nombre de neurones dans la couche j.

Chaque neurone de ce réseau fonctionne en fait comme un perceptron, relié à d'autres perceptrons organisés en forme de couches, et qui transfère le résultat de sa sortie à la couche suivante. En partant de ce principe et en adaptant les notations, le conclusion suivante est trouvée :

[18] Si il y a une seule couche cachée on parle de réseau peu profond (shallow network) et s'il y a plusieurs couches cachées on parle alors de réseau profond (deep network).

[19] Paramètres choisis à la création du modèle par l'utilisateur, ceux-ci ne peuvent être optimisés facilement de manière automatique et nécessitent la création de plusieurs modèles avec différents hyperparamètres dont les résultats seront comparés sur la fonction de perte.

$$a_j^i = \varphi(\sum_k (w_{jk}^i \cdot a_k^{i-1}) + b_j^i)$$

représentant la sortie du neurone j dans la couche i. Si $i = n^{net}$ alors il représente la sortie du réseau, ou sa réponse. La couche de sortie peut être composée d'un seul neurone mais cela n'est pas obligatoire. Si nous cherchons par exemple à estimer à quelle classe une instance appartient nous pouvons représenter chaque classe par un neurone dans cette couche.

Le fonctionement d'un réseau de neurones étant expliqué, il est maintenant nécessaire de comprendre comment un tel réseau est calibré, car sans cette phase de calibration le réseau de neurone serait inefficace. Il existe diverses méthodes permettant d'optimiser un réseau de neurone mais nous ne détaillerons que la "backpropagation" ou rétropropagation, algorithme ayant permi la reprise de la recherche dans le milieu lors de son apparition.

3.3.2.2 Rétropropagation

Avant l'apparition de la rétropropagation, les réseaux de neurones étaient calibrés grâce à la descente du gradient, algorithme consistant à calculer le gradient, la matrice Hessienne et son inverse afin de mettre à jour les paramètres du modèle (poids w_{jk}^i). Cette méthode présente cependant un inconvenient de taille : l'inversion numérique de la matrice Hessienne est très long voir impossible lorsque celle-ci est mal conditionnée. Ces problèmes peuvent être contournés en utilisant un autre algorithme paru en 1986 : la rétropropagation. Pour une plus grande clarté dans la description de cet algorithme, le vecteur contenant les poids w_{jk}^i sera nommé Ω.

La rétropropagation consiste en une série d'instructions répétées T fois, et t sera utilisé en indexe des paramètres pour montrer la valeur d'un paramètre à l'époque[20] t. Le vecteur Ω_t est modifié d'un petit pas dans la direction opposée à celle du gradient à chaque époque pour devenir Ω_{t+1}, et la taille de ce pas est généralement déterminée par une fonction dégressive diminuant au fur et à mesure que t augmente.

[20] Époque : une période ou moment particulier dans l'exécution de l'algorithme.

Algorithm 2 Rétropropagation

1: Attribution de valeurs aléatoires à Ω_0.
2: Choix du pas initial, ρ_0.
3: Choix de la fonction du pas $g(\rho_0, t)$.
4: **begin**
5: For t = 0, ..., T
6: Calcul du gradient : $\nabla R(\Omega_t)$
7: Mise à jour de la taille du pas : $\rho_{t+1} = g(\rho_0, t)$
8: Mise à jour des poids : $\Omega_{t+1} = \Omega_t - \rho_{t+1} \nabla R(\Omega_t)$
9: **end**

TODO : lequel sera utilisé.

3.4 Boosting

Afin d'améliorer les performances de certains algorithmes, de nombreux chercheurs se sont penchés sur les méthodes dites de "boosting". Il s'agit ici plus de "méta-algorithme" que d'algorithme à proprement parler, c'est à dire qu'il s'agit d'une famille d'algorithmes utilisant d'autres algorithmes (apprenants faibles) dans certaines de ses étapes afin d'améliorer les performances que l'apprenant faible choisi aurait eu sans l'utilisation du méta-algortithme. Ces méthodes se basent sur l'entrainement d'apprenants faibles[21] combinés afin de créer un ensemble de prédicteurs considéré comme "fort". Elles se basent souvent sur une règle de décision basée sur les réponses des différents prédicteurs, dans le cas d'une classification binaire cela peut par exemple être représenté par un vote majoritaire de ceux-ci. Avec le boosting, les prédicteurs sont entrainés l'un après l'autre sur un sous-ensemble des données du jeu d'entraînement. Une fois le prédicteur entraîné, toutes les données du jeu d'entraînement sont passées dans celui-ci et l'erreur du prédicteur sur les prédictions est enregistrée. Concrètement c'est la distance entre l'observation et sa prédiction qui est observée, puisque nous tentons avec le boosting de donner l'accent aux données mal prédites pour l'entraînement des prochains prédicteurs. Les probabilités des données les plus incorrectes sont ensuite modifiées afin d'augmenter leur chances d'apparaître dans le sous-ensemble utilisé pour l'entraînement du prochain prédicteur. L'algorithme de boosting le plus connu est probablement AdaBoost, qui tient son nom de "Adaptative boosting"[22], il en existe plusieurs versions à la fois pour la classification et la

[21] Algorithme générant des prédicteurs de relativement mauvaise qualité (mais avec de meilleures performances que le hasard pur), la plupart du temps utilisé en combinaison avec d'autres apprenants faibles afin de créer un ensemble de prédicteurs "fort".

[22] Boosting adaptatif.

régression et c'est ce méta-algorithme qui sera utilisé.

3.4.1 Erreur

En suivant cette méthode nous pouvons réduire l'erreur à chaque nouveau prédicteur ajouté. Cependant il serait naïf de penser que l'on peut utiliser cette méthode infiniment afin d'approcher une erreur nulle, d'abord à cause d'un possible overfit empêchant la généralisation sur le jeu de test, mais surtout car il est possible de démontrer qu'il existe une erreur minimum due aux bruits des données, qui est donc inévitable. Adaboost est donc très sensible au bruit des données comme il est possible de la démontrer :

- N_1: le nombre d'observations du jeu d'entraînement.
- N_2 : le nombre d'observations du jeu de test.
- y_i : la valeur prise dans les données pour l'observation i.
- $y_i^{(t)}$: la valeur réelle exacte pour l'observation i.
- $y_i^{(p)}(x_i)$: la prédiction pour l'observation i.

Définissons l'erreur de prédiction (PE) et l'erreur d'échantillonnage du modèle (ME) :

$$PE = \frac{1}{N_2}\sum_{i=1}^{N_2}[y_i - y_i^{(p)}(x_i)]^2$$

$$ME = \frac{1}{N_2}\sum_{i=1}^{N_2}[y_i^{(t)} - y_i^{(p)}(x_i)]^2$$

Si le bruit peut être considéré comme additif alors :

$$y_i = y_i^{(t)} + n_i$$

où n_i est le bruit de l'observation i. De plus, si nous ajoutons que :

$$E[n] = 0$$
$$E[n_i n_j] = \delta_{ij}\sigma^2$$

Il est possible d'obtenir l'espérance par rapport à (y, x) pour terminer avec :

$$E[PE] = \sigma^2 + E[ME]$$

Ce qui prouve l'existence d'une erreur de prédiction minimum due au bruit et représentée par σ^2. (Drucker)