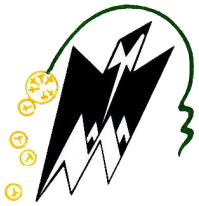
RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITÉ MOULOUD MAMMERI DE TIZI-OUZOU
FACULTÉ DE GÉNIE ÉLECTRIQUE ET INFORMATIQUE
DÉPARTEMENT : INFORMATIQUE



Mémoire de fin d'études

En vue de l'obtention du diplôme de
Master en Informatique

Spécialité : Systèmes Informatiques

Présenté par :
Hacene BELLAHMER

Thème

Implémentation et évaluation d'un modèle d'apprentissage automatique pour l'estimation de la valeur marchande de propriétés immobilières

---

Proposé et dirigé par : Samy SADI

Devant le jury composé de :

M^r. Idir *FILALI*	Président du Jury
M^r. Mohamed *RAMDANI*	Membre du Jury
M^r. Samy *SADI*	Directeur de mémoire

Année universitaire 2019/2020

Dédicaces

Je dédie ce travail à la mémoire de mes grands parents paternelles ainsi que ma tante Aldjia, mon oncle Idir et mon grand père Ouali.

À ma mère, aucun hommage ne pourrait être à la hauteur de tous les sacrifices dont elle a fait preuve.

À ma très chere soeur Mélissa, je tiens à te remercier pour ton aide, ta patience et ton écoute.

À mes chers tantes et oncles, leurs époux et épouses, sans oublier ma chers grand mère à qui je souhaite une longue vie.

À mon frère de coeur Fouad BELHOUAS, à mes très chers camarades de promotion Anis KABLI, Katia IKENE, Syphax BASSAID.

À tous ceux qui on contribué de près ou de loins pour que ce projet soit possible, je vous remercie.

Remerciements

Ce travail n'aurait pas pu voir le jour sans l'aide et l'encadrement du D^r Samy SADI, je le remercie vivement pour la qualité de on encadrement, pour sa patience, sa rigeur et sa disponibilité durant l'élaboration de ce mémoire.

Qu'il trouve ici le témoignage de ma profonde gratitude.

J'adresse mes remerciements les plus sincères aux membres du jury qui ont bien voulu examiné ce modèste travail.

Mes remerciements s'adressent également à tous nos professeurs pour leur generosité et la patience dont ils ont su faire preuve malgré leurs charges professionnelles.

Enfin, je tiens à adresser mes remerciement les plus chaleureux vont à l'égard de ma mére et de ma soeur Mélissa qui m'ont apporté leurs soutiens moral et intelectuel tout au long de mon cursus.

En vous souhaitant une agréable lecture,

Hacene BELLAHMER.

Résumé

L'industrie immobilière est une industrie de base importante pour le développement de l'économie nationale. En effet, Le prix de l'immobilier est un indicateur clé du fonctionnement du marché immobilier et la prédiction du prix de biens immobiliers permet d'aider différentes parties prenantes intervenant dans cette industrie. Plusieurs recherches on proposer des méthodes de prédiction immobilière est cela on utilisant des méthodes de régression simple. Avec l'arrivé du big data et les progrès de l'apprentissage automatique il est maintenant possible de construire des modèles plus sophistiqué afin d'exploiter les informations contenues dans des collections de données immobilière.

Dans le carde de ce mémoire, nous avons dans un premier temps présenté un état de l'art de l'apprentissage automatique et de l'apprentissage par réseaux de neurones, en particulier, pour ce qui est des problèmes de régressions tel que la prédiction des prix de biens immobilier. Dans un second temps, nous avons présenté différentes techniques de prétraitement à effectuer sur les collections de données immobilières ainsi que trois types de modèles proposés pour l'estimation immobilière. Dans un troisième temps, nous implémentons les modèles proposés en commençant par préparer la collection de données choisie. Pour finir, nous comparons les résultats obtenus par nos modèles et nous constatons clairement que le modèle à base de réseaux de neurones obtient de meilleures estimations comparant aux deux autres modèles.

Mots clés : Apprentissage Automatique, Réseaux de Neurones, Évaluation Immobilière, Régression, Bagging, Boosting.

Abstract

The real estate industry is an important basic industry for the development of the national economy. Indeed, the price of real estate is a key indicator of the functioning of the real estate market and the prediction of real estate prices helps different stakeholders involved in this industry. Several researches have proposed methods of real estate prediction is that using simple regression methods. With the advent of big data and advances in machine learning it is now possible to build more sophisticated models to exploit the information contained in real estate data collections.

As part of this thesis, we first presented a state of the art of machine learning and neural network learning, in particular, for regression problems such as the prediction of property prices. In a second step, we presented different pre-processing techniques to be performed on real estate data collections as well as three types of models proposed for real estate estimation. In a third step, we implement the proposed models by first preparing the chosen data collection. Finally, we compare the results obtained by our models and we clearly see that the neural network-based model obtains better estimates compared to the other two models.

Key-words : Machine Learning, Neural Networks, Real Estate Evaluation, Regression, Bagging, Boosting.

Table des matières

Table des figures

Introduction générale

Contexte du travail

Avec l'amélioration du niveau de vie, la demande de biens immobiliers augmente rapidement, tandis que les gens sont de plus en plus préoccupés par le prix des maisons. Les activités économiques liées à l'immobilier sont devenues de plus en plus fréquentes. L'évaluation des biens immobiliers consiste à estimer le prix d'un bien immobilier à un moment donné par une approche scientifique fondée sur une analyse complète des facteurs connexes. L'évaluation des biens immobiliers joue un rôle de plus en plus important dans l'évaluation des taxes foncières, les ventes de biens immobiliers, l'évaluation des prix de location et le contrôle des risques liés aux hypothèques immobilières.

Problématique

La problématique principale traitée dans ce mémoire est celle de l'estimation de la valeur de propriétés immobilières. Pour résoudre cette problématique de régression, nous avons opté pour la création de trois types de modèles différent. Tout d'abord, nous allons définir un modèle utilisant la technique d'apprentissage ensembliste *Bagging* et cela en utilisant l'algorithme du *Random Forest*, par la suite, nous allons construire un second modèle et cela en utilisant une autre technique d'apprentissage ensembliste *Boosting*. Pour le troisième et dernier modèle, nous avons opté pour l'utilisation des réseaux de neurones vus l'intérêt grandissant qu'ils portent actuellement à la communauté scientifique. Ainsi, il faut pouvoir définir un modèle à base de réseaux de neurones efficace pouvant obtenir des résultats encouragent sur l'estimation de ces biens immobiliers, pour qu'à la fin nous puissions comparer les résultats obtenus avec ceux des modèles précèdent.

Contribution

Notre contribution se situe à plusieurs niveaux :

- Premièrement, nous donnons une présentation globale de l'apprentissage automatique ainsi que par réseaux de neurone, notamment pour ce qui est des problèmes de régressions tels que l'estimation de biens immobiliers.
- Deuxièmement, nous réalisons trois modèles effectuant l'estimation de la valeur marchande de propriétés immobilières. En particulier, la présentation des prétraitements nécessaires à effectuer sur les données d'entrée de ces modèles.
- Enfin, la création, l'entraînement et le teste de ses trois modèles sur la collection de donnèes choisie afin de pouvoir ainsi comparer leurs précisions.

Organisation du mémoire

Ce mémoire est organisé en quatre chapitres en plus de l'introduction et de la conclusion générale, les deux premiers chapitres sont théoriques, les deux derniers chapitres quand à eux décrivent les travaux éffectués pour la réalisation des trois modèles proposés.

Le premier chapitre contient une présentation globale sur l'apprentissage automatique, commençant par un bref historique, les types d'apprentissage, les différents algorithmes utilisés en particulier pour ce qui est des problèmes de régression. Par la suite, le second chapitre présente l'utilisation des réseaux de neurones dans le domaine de l'apprentissage automatique en particulier dans le domaine de l'apprentissage profond, en présentant quelques définitions, les différentes couches d'un réseau de neurones ainsi que les différentes fonctions intervenant dans ces réseaux.

Le troisième chapitre comporte une présentation des trois modèles proposés pour l'estimation de la valeur marchande de propriétés immobilières. Par la suite, le quatrième chapitre présente l'implémentation et les résultats des trois modèles, en décrivant tout d'abord, les différentes bibliothèques utilisées.

Chapitre 1

Généralités sur l’apprentissage automatique

1 Introduction

L'apprentissage automatique est un sous-domaine de l'informatique qui s'intéresse à la construction d'algorithmes qui, pour être utile, s'appuie sur un ensemble d'exemples de certains phénomènes. Ces exemples peuvent provenir de la nature, être fabriqués à la main par l'homme ou générés par un autre algorithme. L'apprentissage automatique peut également être défini comme le processus de résolution d'un problème pratique par 1) la collecte d'un ensemble de données, et 2) la construction algorithmique d'un modèle statistique basé sur cet ensemble de données. Ce modèle statistique est censé être utilisé d'une manière ou d'une autre pour résoudre le problème pratique.
Une définition un peu plus générale :[L'apprentissage automatique est le domaine d'étude qui donne aux ordinateurs la possibilité d'apprendre sans être explicitement programmés.][11] Et une autre plus technique : [on dit qu'un programme informatique apprend de l'expérience E en ce qui concerne une tâche T et une mesure de performance P, si sa performance sur T, mesurée par P, s'améliore avec l'expérience E.][12]

Dans ce mémoire, nous nous intéressons à l'estimation des valeurs des propriétés immobilières, c'est pour cela que nous devons nous demander quelles sont les grandes lignes de l'apprentissage automatique ?

Dans ce chapitre, nous présentons un état de l'art de l'apprentissage automatique. Tout d'abord, nous donnons un bref aperçu historique allant des prémisses à l'essor de ce domaine. Ensuite, nous donnons ses différents types et certains algorithmes utilisés. Avant de conclure ce chapitre, nous décrivons la manière de tester et de valider un modèle d'apprentissage automatique.

2 La renaissance de l'apprentissage automatique

En 2006, Geoffrey Hinton et al. ont publié un article[13] montrant comment former un réseau neuronal profond capable de reconnaître des chiffres manuscrits avec une précision de pointe (>98%). Ils ont baptisé cette technique « *Deep Learning* »(apprentissage profond). La formation d'un réseau neuronal profond était largement considérée comme impossible à l'époque et la plupart des chercheurs avaient abandonné l'idée depuis les années 1990. Cet article a ravivé l'intérêt de la communauté scientifique et, très vite, de nombreux nouveaux articles ont démontré que l'apprentissage profond était non seulement possible, mais qu'il permettait d'obtenir des résultats époustouflants qu'aucune autre technique d'apprentissage automatique ne pouvait espérer égaler (grâce à une puissance de calcul énorme et à de grandes quantités de données). Cet enthousiasme s'est rapidement étendu à de nombreux autres domaines de l'apprentissage automatique.

En 10 ans, l'apprentissage automatique a conquis l'industrie : il est aujourd'hui au cœur de la magie des produits de haute technologie, classant les résultats des recherches sur le web, alimentant la reconnaissance vocale des smartphones et recommandant des vidéos, battant le champion du monde au jeu de Go etc.

3 Types d'apprentissage automatique

Il existe de nombreux types de systèmes d'apprentissage automatique. Dans ce qui suit, nous les classons selon qu'ils nécessitent ou non une supervision humaine (supervisés, non supervisés, semi-supervisée, et apprentissage de renforcement)

3.1 Apprentissage supervisé

Dans l'apprentissage supervisé, les données de formation que l'on fournie à l'algorithme comprennent les solutions, appelées étiquètes (ou *labels* en anglais).
Une tâche d'apprentissage supervisée typique est la classification. Le filtre anti-spam en est un bon exemple : il est formé avec de nombreux exemples d'e-mails avec leur classe (spam ou ham), et il doit apprendre à classer les nouveaux e-mails, comme présenté dans la figure 1.1.

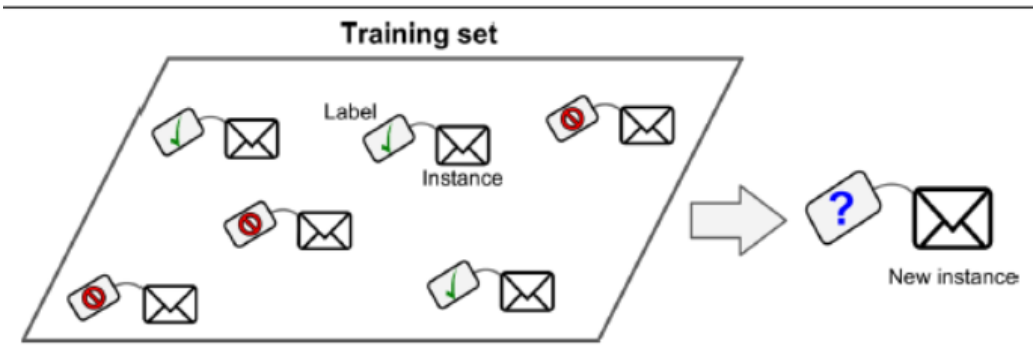


FIGURE 1.1 – Un ensemble de formation labellisé pour l'apprentissage supervisé (par exemple, la classification de spam)[1]

Une autre tâches typique consiste à prédire une valeur numérique cible, tel que le prix d'une voiture, en fonction d'un ensemble de caractéristiques (kilométrage, âge, marque, etc.) appelées prédicteurs. Ce type de tâche est appelé régression. Pour entraîner le système, nous devons lui donner de nombreux exemples de voitures, y compris leurs prédicteurs et leurs étiquettes (c'est-à-dire leurs prix). la figure 1.2 décrit une tache de régression[1]

FIGURE 1.2 – Régression[1]

Noter que certains algorithmes de régression peuvent également être utilisés pour la classification, et vice-versa. Par exemple, la régression logistique est couramment utilisée pour la classification, car elle peut produire une valeur qui correspond à la probabilité d'appartenir à une classe donnée (par exemple, 20% de chances d'être un spam).
Voici quelques-uns des algorithmes d'apprentissage supervisé les plus importants :

- K plus proches voisins ;
- Régression linéaire ;
- Régression logistique ;
- Machines à vecteurs de support (SVM) ;
- Arbres de décision et forêts aléatoires.

3.2 Apprentissage non supervisé

Dans l'apprentissage non supervisé, les données d'apprentissage ne sont pas étiquetées. Le système tente d'apprendre sans professeur.

Le modèle n'a pas de « réponses » dont il peut tirer des enseignements ; il doit donner un sens aux données en fonction des observations elles-mêmes.

L'apprentissage non supervisé nous permet d'aborder les problèmes avec peu ou pas d'idée de ce à quoi nos résultats devraient ressembler. Nous pouvons obtenir une structure à partir de données dont nous ne connaissons pas nécessairement l'effet des variables.
Voici quelques-uns des algorithmes d'apprentissage non supervisé les plus importants :

- Clustering

  :K-Means ; Analyse des clusters hiérarchiques (HCA) ; Maximisation des attentes.

- Visualisation et réduction de la dimensionnalité :
  - Analyse en composantes principales (ACP) ;
  - Kernel PCA ;
  - L'encastrement linéaire local (LLE) ;
  - T-distribué Stochastic Neighbor Embedding (t-SNE).
- Apprentissage des règles d'association :
  - Apriori ;
  - Eclat.

3.2.1 Clustering

Par exemple, disons que nous avons beaucoup de données sur les visiteurs de notre blog. Nous pouvons utiliser un algorithme de *Clustering* pour essayer de détecter des groupes de visiteurs similaires. À aucun moment on dit à l'algorithme de *Clustering* à quel groupe appartient un visiteur : il trouve ces connexions sans notre aide. Par exemple, l'algorithme pourrait remarquer que 40 % de vos visiteurs sont des hommes qui aiment les bandes dessinées et qui lisent généralement votre blog le soir, tandis que 20 % sont de jeunes amateurs de science-fiction qui visite le site durant le week-end, etc. Si nous utilisons un algorithme de Clustering hiérarchique, celui ci peut subdiviser chaque groupe en plus petits groupes. Cela peut aider à cibler des messages pour chaque groupe.

Dans le graphique suivant « figure 1.3 », nous montrons comment un ensemble de points peut être classé pour former trois sous-ensembles :

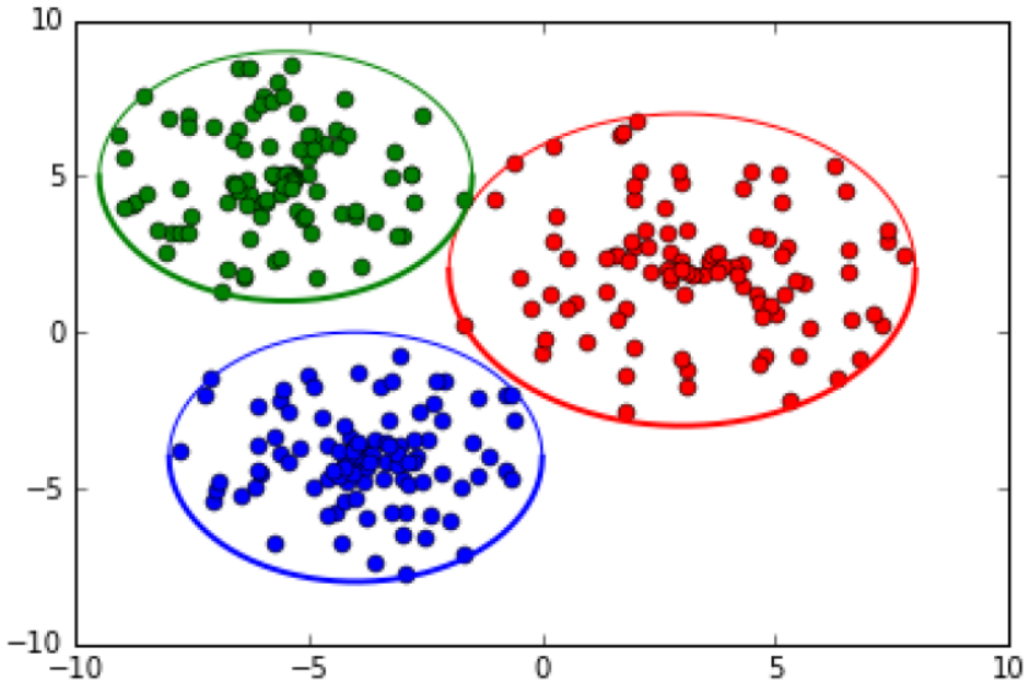


FIGURE 1.3 – Clustering [2]

3.2.2 Réduction de la dimensionnalité

L'objectif est de simplifier les données sans perdre trop d'informations. Une façon d'y parvenir est de fusionner plusieurs caractéristiques corrélées en une seule. Par exemple, le kilométrage d'une voiture peut être très corrélé avec son âge, de sorte que l'algorithme de réduction de la dimensionnalité les fusionnera en une seule caractéristique qui représente l'usure de la voiture. C'est ce qu'on appelle l'extraction de caractéristiques.

3.2.3 La détection des anomalies

La détection des anomalies est un domaine passionnant, qui vise à identifier les objets éloignés qui sont déviants de la distribution générale des données. La détection des valeurs aberrantes s'est avérée essentielle dans de nombreux domaines, par exemple, la détection des transactions inhabituelles par carte de crédit pour éviter la fraude, la détection des défauts de fabrication ou la suppression automatique des valeurs aberrantes d'un ensemble de données avant de les transmettre à un autre algorithme d'apprentissage. Le système est entraîné avec des instances normales et, lorsqu'il voit une nouvelle instance, il peut dire si elle ressemble à une instance normale ou s'il s'agit probablement d'une anomalie, comme le montre le graphe suivant « figure 1.4 » :

FIGURE 1.4 – Détection d'anomalies

3.2.4 Apprentissage des règles d'association

L'objectif est de fouiller dans de grandes quantités de données et de découvrir des relations intéressantes entre les attributs. Par exemple, supposons que nous sommes propriétaire d'un supermarché. L'application d'une règle d'association sur les registres de vente peut révéler que les personnes qui achètent de la sauce barbecue et des pommes de terre ont également tendance à acheter du steak. Par conséquent, on pourrait peut-être placer ces articles à proximité les uns des autres.

3.3 Apprentissage semi-supervisé

Les algorithmes d'apprentissage semi-supervise peuvent traiter des données d'entraînement partiellement étiquetées, généralement beaucoup de données non étiquetées et un peu de données étiquetées et font des prédictions pour tous les points non vus. L'apprentissage semi-supervisé est courant dans les contextes où les données non étiquetées sont facilement accessibles mais où les étiquettes sont coûteuses à obtenir. Différents types de problèmes survenant dans les applications, y compris les tâches de classification, de régression ou de classement, peuvent être présentés comme des exemples d'apprentissage semi-supervisé. L'espoir est que la distribution de données non étiquetées accessibles

à l'algorithme puisse l'aider à obtenir de meilleures performances que dans le cadre de l'apprentissage supervisé. L'analyse des conditions dans lesquelles cela peut être réalisé fait l'objet de nombreuses recherches théoriques et appliquées modernes sur l'apprentissage automatique[14].

Certains services d'hébergement de photos, tels que Google Photos, en sont de bons exemples. Lorsque l'on télécharge des photos de famille sur le service, celui-ci reconnaît automatiquement que la même personne A apparaît sur les photos 1, 5 et 11, tandis qu'une autre personne B apparaît sur les photos 2, 5 et 7. Il s'agit de la partie non supervisée de l'algorithme (Clustering). Maintenant, tout ce dont le système a besoin, c'est que nous lui disons qui sont ces personnes. Une seule étiquette par personne, et il est capable de nommer toutes les personnes présentes sur chaque photo, ce qui est utile pour la recherche de photos.

La plupart des algorithmes d'apprentissage semi-supervisés sont des combinaisons d'algorithmes non supervisés et supervisés. Par exemple, les réseaux de croyances profondes (DBN) sont basés sur des composants non supervisés appelés machines de Boltzmann restreintes (RBM) empilées les unes sur les autres. Les RBM sont formés de manière séquentielle et non supervisée, puis le système entier est réglé avec précision grâce à des techniques d'apprentissage supervisées.

3.4 Apprentissage par renforcement

Le scénario général de l'apprentissage par renforcement est illustré par la figure 1.5. Contrairement au scénario de l'apprentissage supervisé, ici, l'apprenant ne reçoit pas passivement un ensemble de données étiquetées. Au lieu de cela, il recueille des informations par le biais d'un cours d'actions en interagissant avec l'environnement. En réponse à une action, l'apprenant ou l'agent reçoit deux types d'informations : son état actuel dans l'environnement et une récompense à valeur réelle, spécifique à la tâche et à son objectif correspondant. L'objectif de l'agent est de maximiser sa récompense et donc de déterminer le meilleur plan d'action, ou la meilleure politique, pour atteindre cet objectif. Cependant, les informations qu'il reçoit de l'environnement ne sont que la récompense immédiate liée à l'action qu'il vient de mener. Un aspect important de l'apprentissage par renforcement consiste à envisager des récompenses ou des pénalités différées. L'agent est confronté à un dilemme entre l'exploration d'états et d'actions inconnus pour obtenir plus d'informations sur l'environnement et les récompenses, et l'exploitation des informations déjà recueillies pour optimiser sa récompense. C'est ce que l'on appelle le compromis entre l'exploration et l'exploitation, qui est lié au l'apprentissage par renforcement[14].

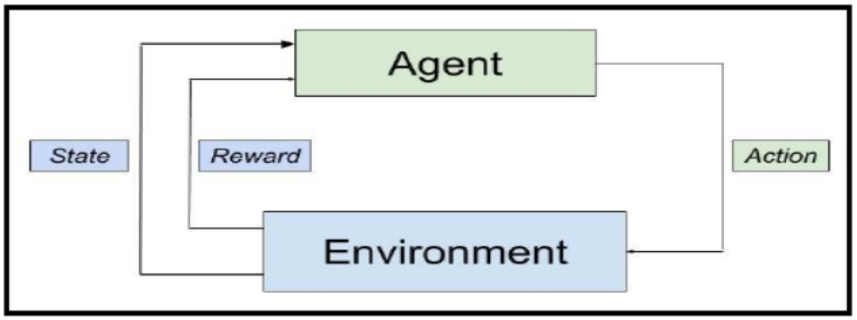


FIGURE 1.5 – Apprentissage par renforcement

Le programme AlphaGo de DeepMind est un bon exemple d'apprentissage par renforcement : il a fait la une des journaux en mars 2016 lorsqu'il a battu le champion du monde Lee Sedol au jeu de Go. Il a appris sa politique gagnante en analysant des millions de parties, puis en jouant de nombreuses parties contre lui-même.

4 Algorithmes d'apprentissage supervisé

4.1 Régression linéaire

La régression linéaire représente une approche très simple pour l'apprentissage supervisé. En particulier, la régression linéaire est un outil utile pour prédire une réponse quantitative. Bien qu'elle puisse sembler un peu ennuyeuse par rapport à certaines approches d'apprentissage statistique plus modernes, la régression linéaire reste une méthode d'apprentissage statistique utile et largement utilisée. En outre, elle constitue un bon point de départ pour de nouvelles approches qui sont considérées comme des généralisations ou des extensions de la régression linéaire[15].

4.1.1 Régression linéaire simple

La régression linéaire simple porte bien son nom : il s'agit d'une approche linéaire très simple permettant de prédire une réponse quantitative Y sur la base d'une seule variable prédictive X. Elle suppose qu'il existe approximativement une relation linéaire entre X et Y. Mathématiquement, nous pouvons écrire cette relation linéaire comme suit :

$$\gamma \approx \beta 0 + \beta 1 \chi \quad (4.1.1)$$

« ≈ » peut être lu comme « est approximativement modelé comme ». Nous décrirons parfois en disant que nous faisons régresser γ sur χ (ou γ sur χ).
Par exemple, χ peut représenter le montant dépensé en publicité télévisée TV et γ peut représenter les ventes. On peut ensuite faire régresser les ventes sur la télévision en ajustant le modèle.

$$\text{Ventes} \approx \beta 0 + \beta 1 \times \text{TV}.$$

Dans l'équation précédente, $\beta 0$ et $\beta 1$ sont deux constantes inconnues qui représentent les termes d'interception et de pente dans le modèle linéaire. Ensemble, $\beta 0$ et $\beta 1$ sont connus comme les coefficients ou paramètres du modèle. Une fois que nous avons utilisé nos données de formation pour produire des estimations $\beta 0$' et $\beta 1$' pour les coefficients du modèle, nous pouvons prévoir les ventes futures sur la base d'une valeur particulière du montant dépensé en publicité télévisée par le calcul :

$$\gamma\text{'} = \beta 0\text{'} + \beta\text{'}\ \text{x},$$

Où γ' indique une prédiction de γ sur la base de χ = x. Nous utilisons ici un symbole, ' , pour indiquer la valeur estimée d'un coefficient de paramètre inconnu, ou pour indiquer la valeur prédite de la réponse. En pratique, $\beta 0$ et $\beta 1$ sont inconnus. Donc, avant de pouvoir utiliser (4.1.1) pour faire des prédictions, nous devons utiliser des données pour estimer les coefficients.

Soit :

$$(x1, y1), (x2, y2), \ldots, (xn, yn)$$

Représentent n paires d'observations, chacune d'entre elles consistant en une mesure de χ et une mesure de γ. Dans l'exemple de la publicité, cet ensemble de données comprend le budget publicitaire de la télévision et les ventes de produits. Notre objectif est d'obtenir des estimations de coefficients $\beta 0$' et $\beta 1$' de sorte que le modèle linéaire (4.1.1) s'adapte bien aux données disponibles, c'est-à-dire que $\gamma' \approx \beta 0' + \beta 1'$xi pour i =1, . . . n. En d'autres termes, nous voulons trouver un point d'intersection $\beta 0$' et une pente $\beta 1$' de telle sorte que la ligne résultante soit aussi proche que possible des points de données. Il existe plusieurs façons de mesurer la proximité. Cependant, l'approche la plus courante consiste à minimiser le critère des moindres carrés. Comme dans l'exemple décrie dans la figure 1.6 :

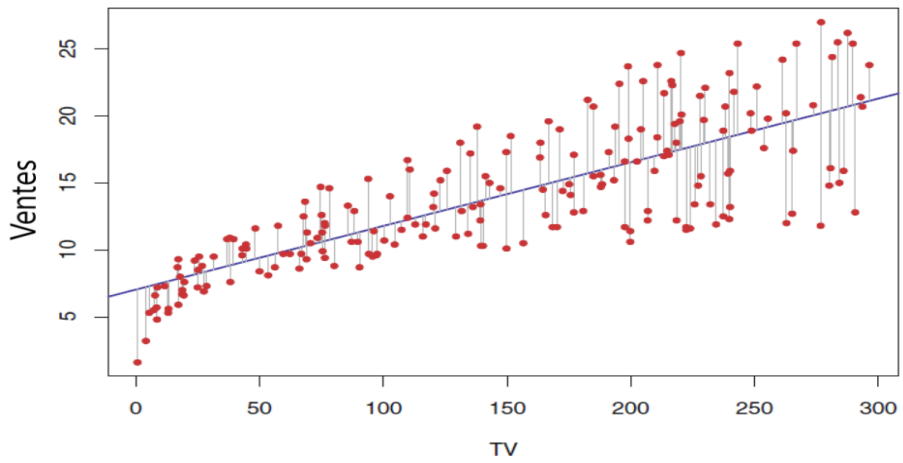


FIGURE 1.6 – Régression linéaire [3]

Pour les données relatives à la publicité, les moindres carrés correspondant à la régression des ventes sur le budget publicitaire de la télévision(TV). L'ajustement est obtenu en minimisant la somme des carrés. Chaque segment de ligne grise représente une erreur, et l'ajustement fait un compromis en faisant la moyenne de leurs carrés. Dans ce cas, un ajustement linéaire saisit l'essence de la relation, bien qu'il soit quelque peu déficient dans la partie gauche du graphique.

Soit $\gamma_i' = \beta_0' + \beta_1' x_i$ est la prédiction pour γ basée sur la ième valeur de χ. Alors $e_i = \gamma_i$-γ_i' représente le ième résidu - c'est la différence entre la ième valeur de réponse observée et la ième valeur de réponse prédite par notre modèle linéaire. Nous définissons la somme résiduelle des carrés (RSS) comme :

$$\text{RSS} = e_1^2 + e_2^2 + \ldots + e_n^2,$$

ou de manière équivalente à :

$$\text{RSS} = (\gamma_1 - \beta_0' - \beta_1' \text{x1})^2 + (\text{y2} - \beta_0' - \beta_1' x_2)^2 + \ldots + (\gamma_n - \beta_0' - \beta_1' x_n)^2.$$

L'approche des moindres carrés choisit β_0' et β_1' pour minimiser le RSS. En utilisant un certain calcul, on peut montrer que les minimiseurs sont :

$$\beta_1' = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sum_{i=1}^{n}(x_i-\bar{x})^2}$$
$$\beta_0' = \bar{y} - \beta_1'\bar{x}$$

où $\bar{y} \equiv \frac{1}{n}\sum_{i=1}^{n} y_i$ et $\bar{x} \equiv \frac{1}{n}\sum_{i=1}^{n} x_i$ sont les moyennes de l'échantillon. En d'autres termes, les formules précédentes définissent les estimations des coefficients des moindres carrés pour la régression linéaire simple.[15]

4.1.2 Régression linéaire multiple

La régression linéaire simple est une approche utile pour prédire une réponse sur la base d'une seule variable prédictive. Cependant, dans la pratique, nous avons souvent plus d'un prédicteur. Par exemple, dans les données sur la publicité, nous avons examiné la relation entre les ventes et la publicité télévisée. Nous disposons également de données sur les sommes dépensées en publicité à la radio et dans les journaux, et nous voudrions peut-être savoir si l'un de ces deux médias est associé aux ventes. Comment pouvons-nous étendre notre analyse des données sur la publicité afin de tenir compte de ces deux prédicteurs supplémentaires ? Une option consiste à effectuer trois régressions linéaires simples distinctes, chacune utilisant un support publicitaire différent comme prédicteur.

Cependant, l'approche consistant à adapter un modèle de régression linéaire simple distinct pour chaque indicateur n'est pas entièrement satisfaisante. Tout d'abord, la manière de faire une prévision unique des ventes étant donné les niveaux des trois budgets publicitaires n'est pas clair, puisque chacun des budgets est associé à une équation de régression distincte. Deuxièmement, chacune des trois équations de régression ignore les deux autres médias pour former les estimations des coefficients de régression. Nous verrons rapidement que si les budgets des médias sont corrélés entre eux dans notre ensemble de données, cela peut conduire à des estimations très trompeuses.

Au lieu d'adapter un modèle de régression linéaire simple et distinct pour chaque prédicteur, une meilleure approche consiste à étendre le modèle de régression linéaire simple afin qu'il puisse s'adapter directement à de multiples prédicteurs. Nous pouvons faire ceci en donnant à chaque prédicteur un coefficient de pente distinct dans un modèle unique. En général, supposons que nous ayons p prédicteurs distincts. Alors, Le modèle de régression linéaire multiple prend la forme :

$$\gamma = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p + e,$$

Comme dans le cas de la régression linéaire simple, les coefficients de régression β_0, β_1, ... β_p sont inconnues et doivent être estimées. Étant donné les estimations β_0', β_1', . . . β_p', nous pouvons faire des prédictions en utilisant la formule :

$$\gamma' = \beta_0' + \beta_1' X_1 + \beta_2' X_2 + ... + \beta_p' X_p,$$

Les paramètres sont estimés en utilisant la même approche des moindres carrés que celle que nous avons vue dans le contexte de la régression linéaire simple. Nous avons choisi β_0, β_1, ... β_p pour minimiser la somme des carrés des résidus :

$$RSS = \sum_{i=1}^{n}(\gamma_i - \gamma_i')^2$$
$$= \sum_{i=1}^{n}(\gamma_i - \beta_0' - \beta_1' x_i 1 - \beta_2' x_i 2 - ... - \beta_p' x_i p)^2$$

La figure 1.7 montre un cadre tridimensionnel, avec deux prédicteurs et une réponse, la ligne de régression des moindres carrés devient un plan. Le plan est choisi pour minimiser la somme des carrés des distances verticales entre chaque observation (indiquée en rouge) et le plan :

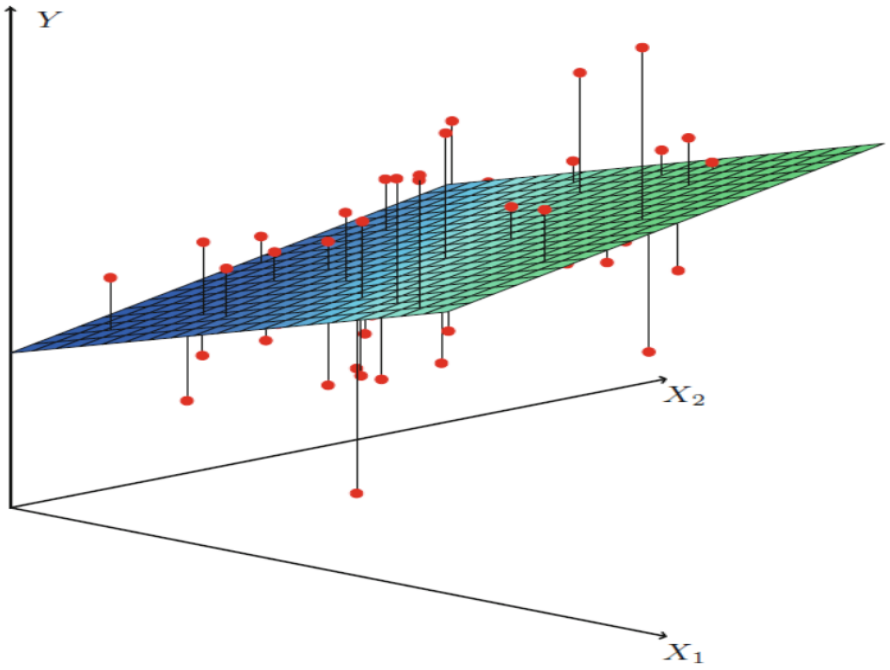


FIGURE 1.7 – Régression linéaire multiple [3]

Dans un cadre tridimensionnel, avec deux prédicteurs et une réponse, la ligne de régression des moindres carrés devient un plan. Le plan est choisi pour minimiser la somme des carrés des distances verticales entre chaque observation (indiquée en rouge) et le plan.

Les valeurs $\beta_0', \beta_1', ...\beta_p'$ sont les estimations du coefficient de régression des moindres carrés multiples. Contrairement aux estimations de régression linéaire simple données, les estimations de coefficients de régression multiples ont des formes un peu compliquées qui sont plus facilement représentées en utilisant l'algèbre matricielle.[15]

4.2 Machines à vecteurs de support (SVM)

Une machine à vecteur de support (SVM) est un algorithme d'apprentissage automatique supervisé qui est principalement utilisé pour la classification. Un SVM tente de trouver un hyperplan, qui sépare les échantillons dans l'ensemble de données. Dans la figure 1.8, nous pouvons voir deux classes de points (rouge et bleu) qui se trouvent dans un espace de caractéristiques bidimensionnel (les axes x et y). Si les deux valeurs x et y d'un point sont inférieures à cinq, le point est bleu. Dans tous les autres cas, le point est rouge. Dans ce cas, les classes sont séparées linéairement, ce qui signifie que nous pouvons les séparer avec un hyperplan.

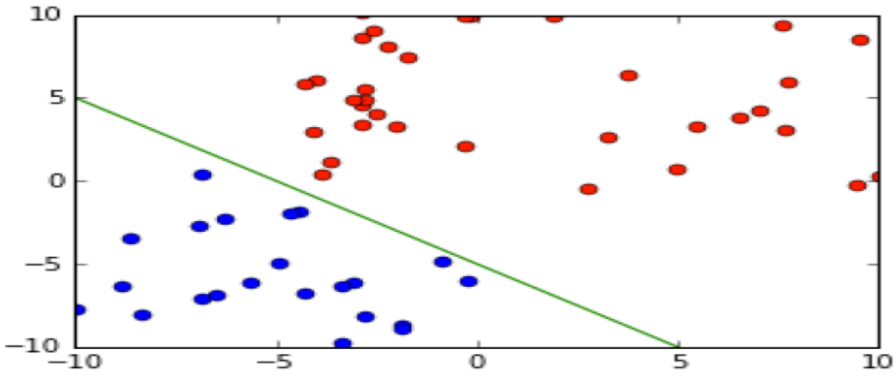


FIGURE 1.8 – SVM

Un SVM essaie de trouver un hyperplan qui maximise la distance entre lui-même et les points. En d'autres termes, parmi tous les hyperplans possibles qui peuvent séparer les échantillons, le SVM trouve celui qui a la distance maximale de tous les points. De plus, les SVM traitent aussi des données qui ne sont pas linéairement séparables. Pour cela, il existe deux méthodes : l'introduction de marges souples ou l'utilisation de l'astuce du noyau.[2]

- Les marges souples fonctionnent en autorisant quelques éléments mal classés tout en conservant la capacité de prédiction la plus élevée de l'algorithme. En pratique, il est préférable de ne pas surapprendre le modèle d'apprentissage de la machine, et nous pourrions le faire en assouplissant certaines des hypothèses de la machine à vecteur de support.

- L'astuce du noyau résout le même problème d'une manière différente. Imaginons que nous ayons un espace à deux dimensions, mais que les classes soient linéairement inséparables. L'astuce du noyau utilise une fonction noyau qui transforme les données en y ajoutant d'autres dimensions. Dans notre cas, après la transformation, les données seront tridimensionnelles. Les classes linéairement inséparables dans l'espace bidimensionnel deviendront linéairement séparables dans les trois dimensions et notre problème est résolu

4.3 Arbres de décision

Comme les SVM, les arbres de décision sont des algorithmes d'apprentissage automatique polyvalents qui peuvent effectuer des tâches de classification et de régression, et même des tâches à sorties multiples. Ce sont des algorithmes très puissants, capables de s'adapter à des ensembles de données complexes. Les arbres de décision sont également les composants fondamentaux des forêts aléatoires, qui comptent parmi les algorithmes d'apprentissage automatique les plus puissants disponibles aujourd'hui[1].

Essentiellement, ils apprennent une hiérarchie de questions « si/autre », menant à une décision. Ces questions sont similaires à celles que l'on peut poser dans un jeu de 20 questions. Imaginons que l'on veut faire la distinction entre les quatre animaux suivants : ours, faucons, pingouins et dauphins. Votre but est d'obtenir la bonne réponse en posant le moins de questions possible. On peut commencer par demander si l'animal à des plumes, une question qui réduit vos possibilités à deux animaux seulement. Si la réponse est « oui », on peut poser une autre question qui pourrait nous aider à faire la distinction entre les faucons et les pingouins. Par exemple, nous pouvons demander si l'animal peut voler. Si l'animal n'a pas de plumes, vos choix d'animaux possibles sont les dauphins et les ours, et nous devrons poser une question pour distinguer ces deux animaux - par exemple, demander si l'animal à des nageoires.

Cette série de questions peut être exprimée sous la forme d'un arbre de décision, comme le montre la figure suivante « figure 1.9 » :

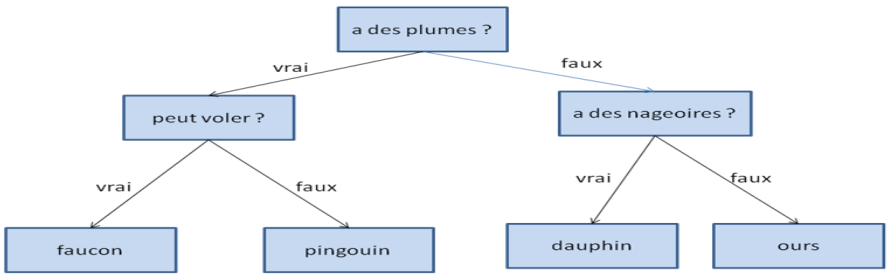


FIGURE 1.9 – Un arbre de décision pour distinguer entre plusieurs animaux

Dans cette illustration, chaque nœud de l'arbre représente soit une question, soit un nœud terminal (également appelé feuille) qui contient la réponse. Les bords relient les réponses à une question à la question suivante que nous poserions.

En langage d'apprentissage automatique, nous avons construit un modèle permettant de distinguer quatre classes d'animaux (faucons, pingouins, dauphins et ours) en utilisant les trois caractéristiques « des plumes », « peut voler » et « a des nageoires ». Au lieu de construire ces modèles à la main, nous pouvons les apprendre à partir de données grâce à un apprentissage supervisé.

Les arbres de décision présentent deux avantages par rapport à de nombreux algorithmes : le modèle résultant peut être facilement visualisé et compris par des non-experts (du moins pour les petits arbres), et les algorithmes sont totalement invariants en ce qui concerne la mise à l'échelle des données. Comme chaque caractéristique est traitée séparément et que les divisions éventuelles des don-

nées ne dépendent pas de l'échelle, aucun prétraitement tel que la normalisation ou la standardisation des caractéristiques n'est nécessaire pour les algorithmes d'arbre de décision. En particulier, les arbres de décision fonctionnent bien lorsque nous disposons de caractéristiques à des échelles complètement différentes, ou d'un mélange de caractéristiques binaires et continues.

Le principal inconvénient des arbres de décision est qu'ils ont tendance à se surajuster et à fournir de mauvaises performances de généralisation. C'est pourquoi, dans la plupart des applications, les méthodes d'ensemble comme les forêts aléatoires dont nous parlons ci-après sont généralement utilisées à la place d'un arbre de décision unique.

4.4 Les forêts aléatoires

Comme nous venons de l'observer, un des principaux inconvénients des arbres de décision est qu'ils ont tendance à se surajuster aux données de formation. Les forêts aléatoires sont un moyen de remédier à ce problème. Une forêt aléatoire est essentiellement une collection d'arbres de décision, où chaque arbre est légèrement différent des autres. L'idée derrière les forêts aléatoires est que chaque arbre peut faire un assez bon travail de prévision, mais qu'il va probablement sur-apprendre une partie des données.Si nous construisons plusieurs arbres, qui fonctionnent tous bien et se surajoutent de différentes façons, nous pouvons réduire le nombre de surajouts en faisant la moyenne de leurs résultats. Cette réduction du sur-ajustement, tout en conservant le pouvoir de prédiction des arbres, peut être démontrée par des calculs mathématiques rigoureux.

Pour mettre en œuvre cette stratégie, nous devons construire de nombreux arbres de décision. Chaque arbre doit permettre de prévoir l'objectif de manière acceptable et doit également être différent des autres arbres. Les forêts aléatoires tirent leur nom de l'injection de hasard dans la construction des arbres pour s'assurer que chaque arbre est différent. Les arbres d'une forêt aléatoire sont randomisés de deux manières : en sélectionnant les points de données utilisés pour construire un arbre et en sélectionnant les caractéristiques de chaque test de division.

Les forêts aléatoires est l'algorithme le plus connu de la technique d'appentissage ensembliste « *bagging* » ou « *bootstrap aggregating* » qui consiste à créer plusieurs entités d'un même modèle (plusieurs arbres de décisions) et d'entrainer chacun de ces arbre sur une portion aléatoire d'une collection de données, après avoir entrainer chaque arbre, nous pouvons regrouper les résultats de chaque arbre afin d'effectuer la prédiction, la figure 1.10 montre la technique du « *bagging* ».

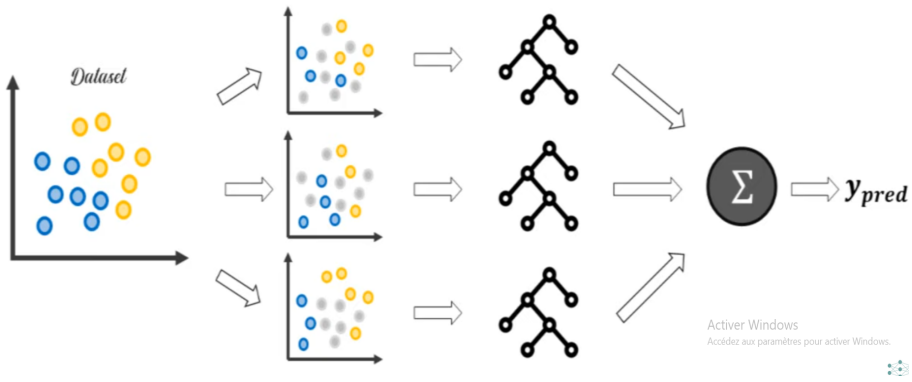


FIGURE 1.10 – Bagging [1]

4.5 Gradient Boosting

Le Boosting fait référence à toute méthode d'Ensemble qui peut combiner plusieurs apprenants faibles en un apprenant fort. L'idée générale de la plupart des méthodes de boosting est d'entraîner les modèles de manière séquentielle, chacun essayant de corriger les érreurs de son prédécesseur, par conséquent nous obtenons des modèles complémentaires. Plusieurs algorithmes de Boosting sont disponibles, l'un d'entre eux est le Gradient Boosting qui fonctionne en ajoutant séquentiellement des modèles à un ensemble, chaque modèle corrigeant son prédécesseur. Cependant, au lieu de modifier les poids des instances à chaque itération, cette méthode tente de faire correspondre le nouveau modèle aux erreurs résiduelles du prédicteur précédent.

Le Gradient Boosting fonctionne très bien avec les tâches de régression. C'est ce qu'on appelle *Gradient Boosted Regression Trees (GBRT)*, lors de l'entrainement d'un GBRT modèle. Afin de trouver le nombre optimale d'arbres, nous pouvons utiliser un Early Stopping afin d'éviter entre autre le sur-apprentissage, la bibliothèque XGBoost permet de mettre en œuvre des méthodes de Gradient boosting, la figure 1.11 montre un modèle GBRT en utilisant un Early Stopping afin de trouver le nombre d'arbres optimale.

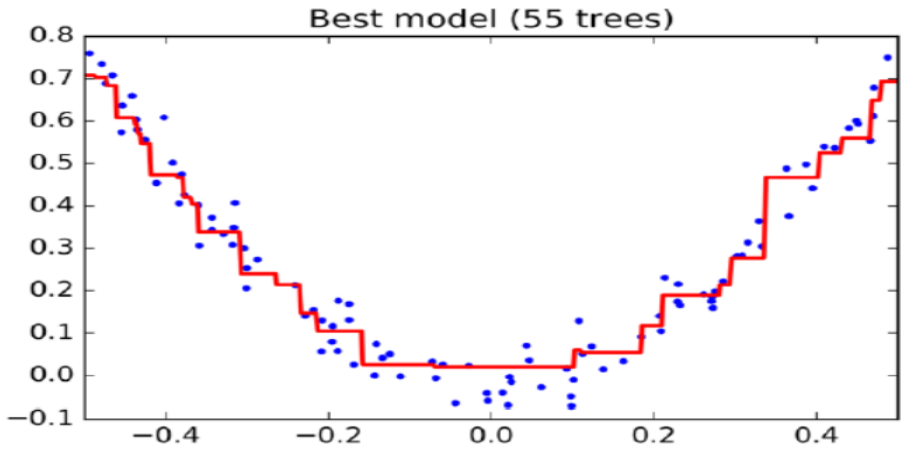


FIGURE 1.11 – GBRT avec Early Stopping [1]

1. https ://machinelearnia.com/

5 Tester et valider un modèle d'apprentissage automatique :

La seule façon de savoir à quel point un modèle se généralisera à de nouveaux cas est de l'essayer réellement sur de nouveaux cas. Une façon d'y parvenir est de mettre le modèle en production et de contrôler ses performances. Cela fonctionne bien, mais si le modèle est horriblement mauvais, les utilisateurs se plaindront - ce n'est pas la meilleure idée. Une meilleure option consiste à diviser les données en deux ensembles : l'ensemble de formation et l'ensemble de test. Comme ces noms l'indiquent, on entraîne le modèle à l'aide de l'ensemble d'entraînement et on le teste à l'aide de l'ensemble de test. Le taux d'erreur sur les nouveaux cas est appelé erreur de généralisation (ou erreur hors échantillon), et en évaluant le modèle sur l'ensemble de test, on obtient une estimation de cette erreur. Si l'erreur de formation est faible (c'est-à-dire que le modèle fait peu d'erreurs sur l'ensemble de formation) mais que l'erreur de généralisation est élevée, cela signifie que le modèle est en surapprentissage sur les données de formation. Il est courant d'utiliser 80 % des données pour la formation et d'en retenir 20 % pour les tests.

L'évaluation d'un modèle est donc assez simple : il suffit d'utiliser un ensemble de tests. Supposons maintenant que l'on hésite entre deux modèles (disons un modèle linéaire et un modèle polynomial) : comment pouvons-nous décider ? Une option consiste à former les deux modèles et à comparer leur généralisation à l'aide de la série de tests.

Supposons maintenant que le modèle linéaire se généralise mieux, mais que l'on veut appliquer une certaine régularisation pour éviter le surajustement. La question est la suivante : comment choisir la valeur de l'hyperparamètre de régularisation ? Une option consiste à former 100 modèles différents en utilisant 100 valeurs différentes pour cet hyperparamètre. Supposons que l'on trouve la meilleure valeur d'hyperparamètre qui produise un modèle avec la plus faible erreur de généralisation, disons une erreur de 5% seulement.

On lance donc ce modèle en production, mais malheureusement il ne fonctionne pas aussi bien que prévu et produit 15% d'erreurs. Qu'est-ce qui vient de se passer ?
Le problème est que l'on a mesuré l'erreur de généralisation plusieurs fois sur l'ensemble de tests, et que l'on a adapté le modèle et les hyperparamètres pour produire le meilleur modèle possible. Cela signifie que le modèle a peu de chances de fonctionner aussi bien sur de nouvelles données.

Une solution commune à ce problème est de disposer d'un deuxième ensemble de blocage appelé ensemble de validation. On entraîne plusieurs modèles avec différents hyperparamètres à l'aide de l'ensemble d'entraînement, on sélectionne le modèle et les hyperparamètres qui donnent les meilleurs résultats dans l'ensemble de validation, et lorsqu'on est satisfait du modèle, nous effectuons un seul test final par rapport à l'ensemble de test pour obtenir une estimation de l'erreur de généralisation.
Pour éviter de « gaspiller » trop de données de formation dans les ensembles de validation, une technique courante consiste à utiliser la validation croisée : l'ensemble de formation est divisé en sous-ensembles complémentaires, et chaque modèle est formé par rapport à une combinaison différente de ces sous-ensembles et validé par rapport aux autres parties. Une fois que le type de modèle et les hyperparamètres ont été sélectionnés, un modèle final est formé en utilisant ces hyperparamètres sur l'ensemble de formation complet, et l'erreur généralisée est mesurée sur l'ensemble de test.

6 Conclusion

Dans ce premier chapitre, nous avons présenté de manière globale les principaux axes de l'apprentissage automatique, certaines définitions, ses types, quelques algorithmes utilisés en particulier en apprentissage supervisé, avant de finir par décrire la manière de tester et de valider un modèle d'apprentissage automatique.

Les réseaux de neurones sont un sujet de recherche depuis plusieurs années, mais les progrès réalisés étaient limités en raison des limites des calculs et des données de l'époque. Lorsque les chercheurs ont atteint le point culminant de l'apprentissage automatique et des réseaux de neurones, le domaine de l'apprentissage profond *Deep Learning* est apparu, encadré par le développement de réseaux de neurones profonds, c'est-à-dire des réseaux de neurones improvisés comportant beaucoup plus de couches. l'apprentissage profond a excellé dans les nouvelles frontières où l'apprentissage automatique prenait du retard. C'est pour cela que le second chapitre est consacré à la présentation de l'apprentissage par réseaux de neurones.

Chapitre 2

Apprentissage par réseaux de neurones

1 Introduction

L'examen de l'architecture du cerveau humain a permit de trouver l'inspiration sur la façon de construire une machine intelligente. C'est l'idée clé qui a inspiré les réseaux de neurones artificiels. Cependant, les réseaux neuronaux artificiels sont devenus progressivement très différents de leurs cousins biologiques. Certains chercheurs affirment même que nous devrions abandonner complètement l'analogie biologique (par exemple, en disant « unités » plutôt que « neurones »), de peur de limiter notre créativité à des systèmes biologiquement plausibles.

Les réseaux de neurones existent depuis de nombreuses années mais récemment, ils ont régulièrement gagné du terrain sur de nombreux autres algorithmes d'apprentissage automatique concurrents. Cette résurgence est due à la rapidité des ordinateurs, à l'utilisation des unités de traitement graphique (GPU) plutôt que des unités de traitement informatique (CPU), à l'amélioration des algorithmes et de la conception des réseaux neuronaux, et à des ensembles de données de plus en plus volumineux.

Comme les réseaux de neurones sont au cœur même de l'apprentissage profond, ces derniers sont utilisés pour régler des tâches de régression, ce qui est utile pour notre problème d'estimation. par conséquent, il est utile de savoir quels sont les différents points de l'apprentissage par réseaux de neurones ?

Nous présentons dans ce chapitre, les axes fondamentaux de l'apprentissage par réseaux de neurones. En premier lieu, il nous a semblé intéressant de faire un parallèle entre les neurones biologiques et artificiels qui sont la base de ses réseaux. En second lieu, nous nous appliquons à distinguer entre les différentes couches d'un réseau de neurones. En troisième lieu, nous présentons ce qu'est une fonction d'activation, son intérêt, ainsi que certaines fonctions d'activations. En quatrième lieu, nous définissons ce qu'est une fonction de perte (*loss function*). Avant de conclure ce chapitre, nous présentons ce qu'est un optimiseur, ainsi que les différents types d'optimiseurs utilisés.

2 Des neurones biologiques aux neurones artificiels

Les réseaux de neurones ont été développés pour simuler le système nerveux humain pour des tâches d'apprentissage automatique en traitant les unités de calcul dans un modèle d'apprentissage d'une manière similaire aux neurones humains. La grande vision des réseaux de neurones est de créer une intelligence artificielle en construisant des machines dont l'architecture simule les calculs dans le système ssnerveux humain.

La nature gourmande en données et en calcul intensif des réseaux de neurones a été considérée comme un obstacle à leur utilisabilité. Finalement, l'augmentation considérable de la puissance de calcul depuis les années 1990 et une plus grande disponibilité des données ont conduit à des succès accrus des réseaux de neurones, et ce domaine a rené sous la nouvelle étiquette de «*deep learning*».

Les réseaux de neurones sont théoriquement capables d'apprendre n'importe quelle fonction mathématique avec suffisamment de données d'entraînement et ils sont devenus des techniques d'apprentissage automatique populaires qui simulent le mécanisme d'apprentissage dans les organismes biologiques. Le système nerveux humain contient des cellules, appelées neurones.

Les neurones sont connectés les uns aux autres grâce à l'utilisation d'axones et de dendrites, et les régions de connexion entre les axones et les dendrites sont appelées synapses[5]. Ces connexions sont illustrées à la figure2.1.

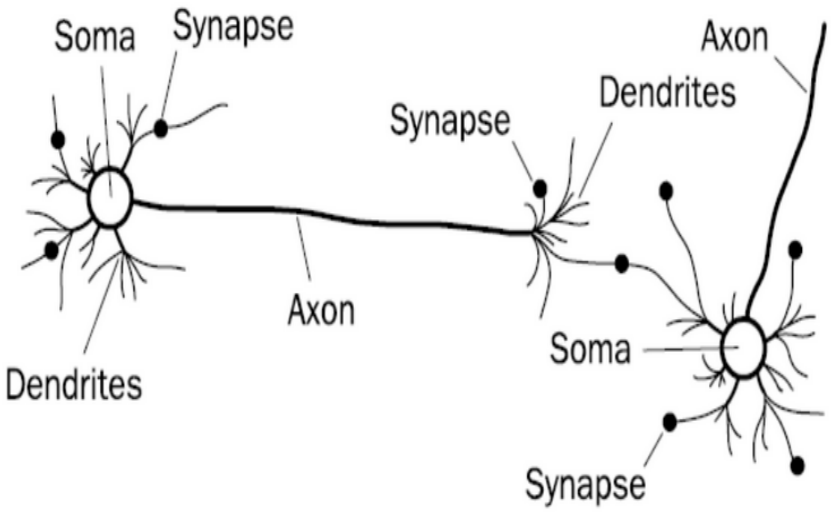


FIGURE 2.1 – Réseau de neurones biologique[4]

Les forces des connexions synaptiques changent souvent en réponse à des stimuli externes. Ce changement est la façon dont l'apprentissage prend place dans les organismes vivants. Ce mécanisme biologique est simulé dans des réseaux de neurones artificiels, qui contiennent des unités de calcul appelées neurones. Les unités de calcul sont reliées les unes aux autres par des poids, qui jouent le même rôle que les forces des connexions synaptiques dans les organismes biologiques. Chaque entrée d'un neurone est mise à l'échelle avec un poids, ce qui affecte la fonction calculée à cette unité[5]. Cette architecture est illustrée à la figure 2.2.

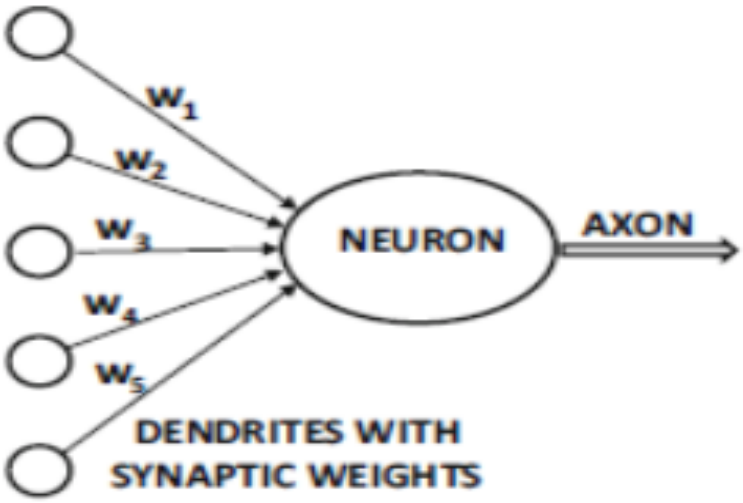


FIGURE 2.2 – Réseau de neurones artificiel[5]

3 Couches d'un réseau de neurones

Un réseau de neurones peut avoir un nombre indéfini de neurones, qui sont organisés en couches interconnectées. La couche d'entrée représente l'ensemble de données et les conditions initiales, la couche de sortie peut avoir plus d'un neurone. Cela est particulièrement utile dans la classification, où chaque neurone de sortie représente une classe. Les couches intermédiaires (entre l'entrée et la sortie) sont appelées couches cachées parce que les calculs effectués ne sont pas visibles pour l'utilisateur. L'architecture spécifique des réseaux neuronaux multicouches est appelée réseaux à propagation avant, car les couches successives s'alimentent les unes les autres dans le sens de l'entrée vers la sortie. L'architecture par défaut des réseaux à propagation avant suppose que tous les nœuds d'une couche sont connectés à ceux de la couche suivante.

Le figure 2.3 montre les différentes couches d'un réseaux de neurones :

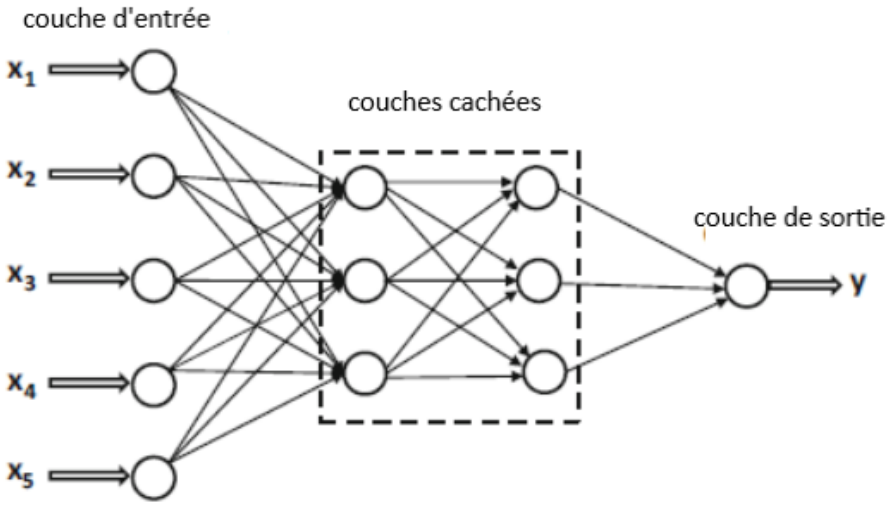


FIGURE 2.3 – Réseaux de neurones multicouches[5]

4 Fonctions d'activations

Une fonction d'activation est la fonction qui prend l'entrée combinée z décrite dans la figure 2.4, lui applique une fonction et lui transmet la valeur de sortie, essayant ainsi d'imiter la fonction d'activation/désactivation. La fonction d'activation détermine donc l'état d'un neurone. Si les neurones n'ont pas de fonctions d'activation, leur sortie serait la somme des poids des entrées, $\sum_i \omega_i \chi_i$, ce qui est une fonction linéaire, et l'ensemble du réseau de neurones, c'est-à-dire une composition de neurones, deviendrait une composition de fonctions linéaires, ce qui est également une fonction linéaire. Cela signifie que même si nous ajoutons des couches cachées, le réseau sera toujours équivalent à un simple modèle de régression linéaire, avec toutes ses limites. Pour transformer le réseau en une fonction non linéaire, nous utiliserons des fonctions d'activation non linéaires pour les neurones. Habituellement, tous les neurones d'une même couche ont la même fonction d'activation, mais différentes couches peuvent avoir des fonctions d'activation différentes[2].

Il existe plusieurs choix de fonctions d'activation, les plus courantes étant la fonction sigmoïde et la ReLU « *rectified linear unit* ».

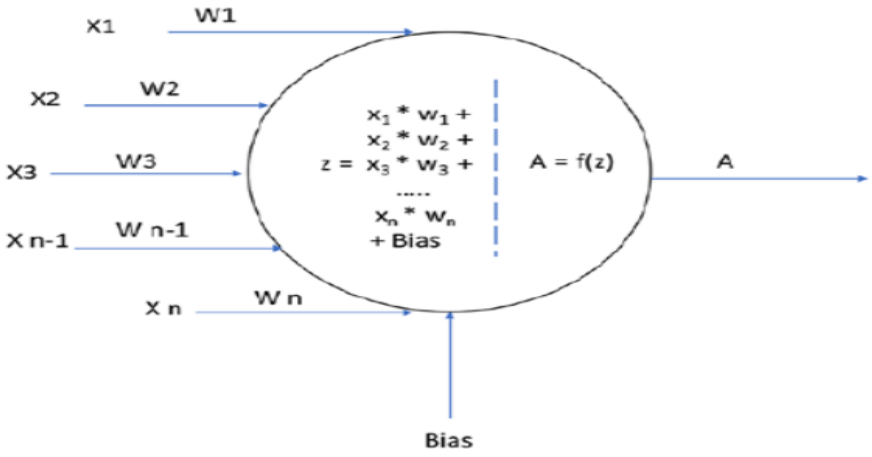


FIGURE 2.4 – Fonction d'activation sur un neurone unique[6]

4.1 Fonction d'activation linéaire

La fonction d'activation la plus élémentaire est l'activation identitaire ou linéaire,

$$f(\chi) = \chi$$

La fonction d'activation linéaire est souvent utilisée dans le nœud de sortie, lorsque la cible est une valeur réelle. Elle est même utilisée pour les sorties discrètes lorsqu'une fonction de perte de substitution lissée doit être mise en place. Le graphe suivant « figure 2.5 » montre le tracé d'une fonction d'activation linéaire.

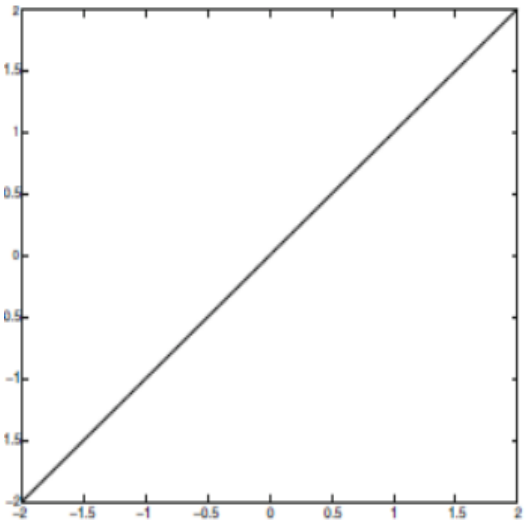


FIGURE 2.5 – Fonction d'activation linéaire

4.2 Fonction d'activation signe

La fonction d'activation signe « figure 2.6 » peut être utilisée pour établir une correspondance avec des sorties binaires au moment de la prédiction, sa non-différentiabilité empêche son utilisation pour créer la fonction de perte au moment de l'entraînement. Par exemple, alors que le perceptron utilise la fonction de signe pour la prédiction, le critère du perceptron à l'entraînement ne nécessite qu'une activation linéaire.

$$f(\chi) = sign(\chi)$$

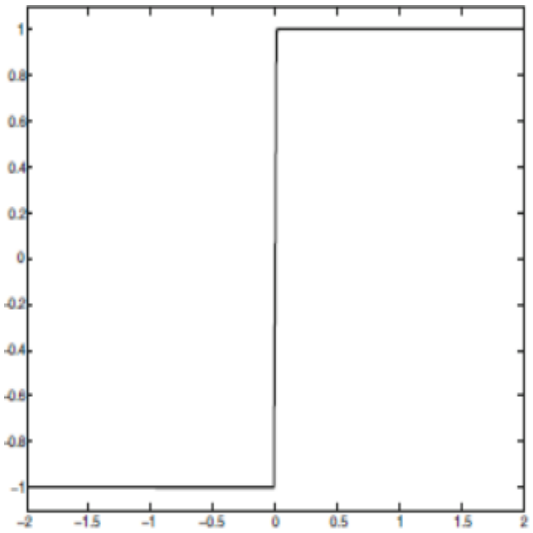


FIGURE 2.6 – Fonction d'activation signe

4.3 Fonction sigmoïde

Une fonction sigmoïde « figure 2.7 » est définie comme suit :

$$f(\chi) = \frac{1}{1+e^{-\chi}}$$

Elle rend la sortie entre 0 et 1, comme le montre l'illustration suivante. La sortie non linéaire (en forme de s comme indiqué) améliore très bien le processus d'apprentissage, car elle ressemble beaucoup au principe suivant - faible influence : sortie faible et influence élevée : sortie élevée - et limite également la sortie dans une fourchette de 0 à 1.

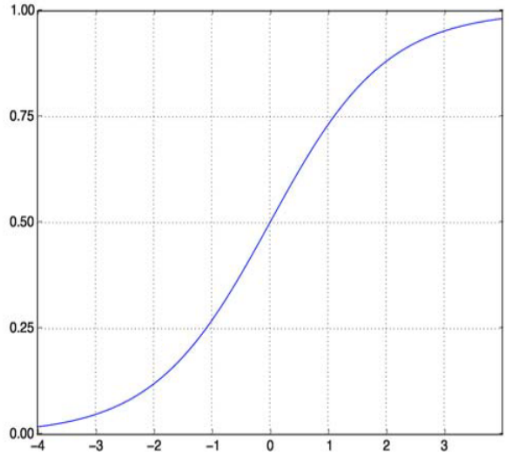


FIGURE 2.7 – Fonction sigmoïde [7]

4.4 Tangente hyperbolique

La fonction tangente hyperbolique « figure 2.8 » tanh a une forme similaire à celle de la fonction sigmoïde, sauf qu'elle est rééchelonnée horizontalement et traduite/rééchelonnée verticalement à [-1, 1], Les fonctions tanh et sigmoïde sont liées comme suit :

$$f(\chi) = 2 * sigmoid(2\chi) - 1$$

La fonction tanh est préférable à la fonction sigmoïde lorsque les résultats des calculs doivent être à la fois positifs et négatifs. De plus, son centrage moyen et son gradient plus important (en raison de l'étirement) par rapport au sigmoïde facilitent l'entraînement. Les fonctions sigmoïde et tanh ont été les outils historiques de choix pour intégrer la non-linéarité dans le réseau de neurones.

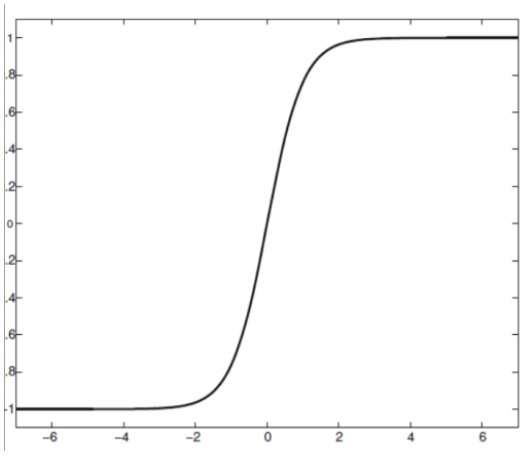


FIGURE 2.8 – Tangente hyperbolique

4.5 ReLU et Leaky ReLU

ReLU «*rectified linear unit*» « figure 2.9 » utilise la fonction f(z) = max(0,z), ce qui signifie que si la sortie est positive, elle produira la même valeur, sinon elle produira 0. La plage de sortie de la fonction est illustrée dans le visuel suivant :

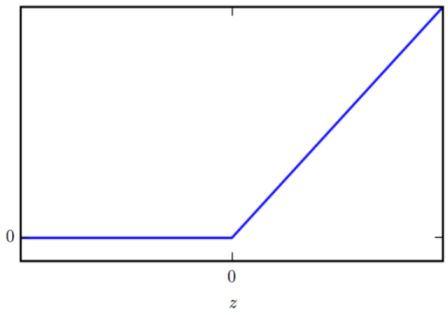


FIGURE 2.9 – ReLU [8]

La fonction peut sembler linéaire, mais elle ne l'est pas. ReLU est une fonction non linéaire valide et fonctionne en fait très bien comme fonction d'activation. Elle n'améliore pas seulement les performances, mais elle permet de réduire considérablement le nombre de calculs pendant la phase de formation. C'est le résultat direct de la valeur 0 dans la sortie lorsque z est négatif, ce qui désactive le neurone, mais en raison de la ligne horizontale avec 0 comme sortie, nous pouvons parfois être confrontés à de sérieux problèmes. avec cette ligne horizontale, qui est une constante avec une dérivée de 0 et qui peut donc devenir un goulot d'étranglement pendant l'entraînement, car les poids seront mis à jour régulièrement. Pour contourner le problème, une nouvelle fonction d'activation a été proposée : Le leaky ReLU « figure 2.10 », où la valeur négative produit une ligne légèrement inclinée au lieu d'une ligne horizontale, ce qui permet de mettre à jour les poids par rétropropagation de manière efficace.

leaky ReLU est définie comme :

- $f(z) = z$; lorsque $z > 0$.
- $f(z) = \propto z$; lorsque $z < 0$ et où $\propto$ est un paramètre qui est définie comme une petite constante, disons 0,005.

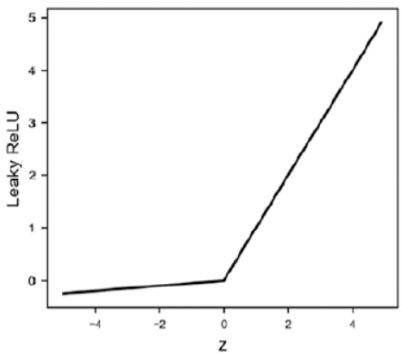


FIGURE 2.10 – leaky ReLU

5 Fonction de perte

La fonction de perte *loss function* est la mesure qui aide un réseau à comprendre s'il apprend dans la bonne direction. Pour formuler la fonction de perte en termes simples, on peut considérer la note obtenu par un étudiant lors d'un examen. Supposons que cet étudiant se soit présenté à plusieurs tests sur le même sujet : quelle est la métrique à utiliser pour comprendre la performance pour chaque test ? Évidemment, le score du test. Supposons que cette étudiant à obtenu 56, 60, 78, 90 et 96 sur 100 dans cinq tests consécutifs, on peut constater clairement que l'amélioration des résultats est une indication des performances. De même, comment un réseau peut-il savoir s'il améliore son processus d'apprentissage à chaque itération ? Il utilise la fonction de perte, qui est analogue à la note de l'examen. La fonction de perte mesure essentiellement la perte par rapport à la cible. Supposons que l'on développe un modèle pour prédire si un élève va réussir ou échouer et que la chance de réussite ou d'échoué soit définie par la probabilité. Ainsi, 1 indiquerait qu'il réussira avec une certitude de 100 % et 0 indiquerait qu'il échouera définitivement. Le modèle tire les leçons des données et prédit un score de 0,87 pour que l'élève réussisse. La perte réelle serait donc ici de 1,00 - 0,87 = 0,13. S'il répète l'exercice avec quelques mises à jour des paramètres afin de s'améliorer et obtient maintenant une perte de 0,40, il comprendrait que les changements qu'il a apporté n'aident pas le réseau à apprendre de manière appropriée.

En fonction du type de résultat des données, nous avons plusieurs fonctions de perte standard définies dans l'apprentissage automatique. Pour les cas d'utilisation de régression, en voici quelques-unes :

- **Mean Squared Error (MSE) :** Différence moyenne au carré entre la valeur réelle et la valeur prévue. Le carré de la différence permet de pénaliser facilement le modèle pour une différence plus élevée. Ainsi, une différence de 3 entraînerait une perte de 9, mais une différence de 9 entraînerait une perte de 81.

- **Mean Absolute Error (MAE) :** L'erreur moyenne absolue entre la valeur réelle et la valeur prédite.

Pour les résultats catégoriques, la prédiction serait pour une classe, par exemple si un élève réussira (1) ou échouera (0), si le client fera un achat ou non, si le client sera en défaut de paiement ou non, et ainsi de suite. Certains cas d'utilisation peuvent avoir plusieurs classes comme résultat, par exemple en classant les types de maladies (type A, B ou C), en classant les images comme des chats, des chiens, des voitures, des chevaux, des paysages, etc... Dans ces cas, les pertes définies dans les cas précédents ne peuvent pas être utilisées pour des raisons évidentes. Il faudrait quantifier le résultat de la probabilité de la catégorie et définir les pertes en fonction des estimations de la probabilité. Voici quelques choix populaires de pertes pour les résultats catégoriques :

- **Binary cross-entropy :** Définit la perte lorsque les résultats catégoriques sont une variable binaire, c'est-à-dire avec deux résultats possibles : (réussite/échec) ou (oui/non), généralement utilisé pour des modèles de régression logistique[16].

- **Categorical cross-entropy :** Définit la perte lorsque les résultats de la catégorie sont non binaires, c'est-à-dire >2 résultats possibles : (Oui/Non/Peut-être) ou (Type 1/ Type 2/... Type n).

Dans un réseau de neurone, la fonction de perte prend les prédictions du réseau et la cible réelle et calcule un score de distance, en capturant la performance du réseau, comme le montre la « figure2.11 »[9].

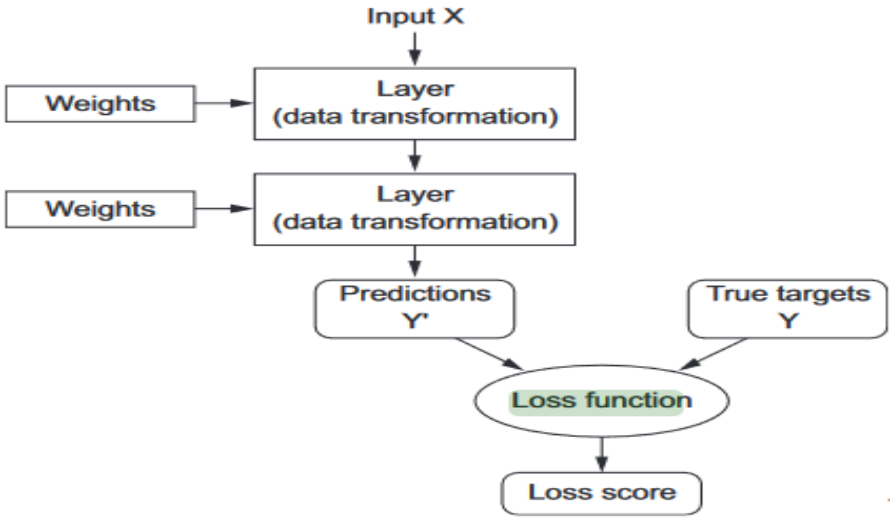


FIGURE 2.11 – La fonction de perte mesure la qualité de la sortie du réseau de neurone[9]

Dans le graphique suivant « figure 2.12 », nous pouvons voir comment la fonction de perte diminue à chaque époque. Cela montre comment le réseau apprend progressivement sur les données de formation :

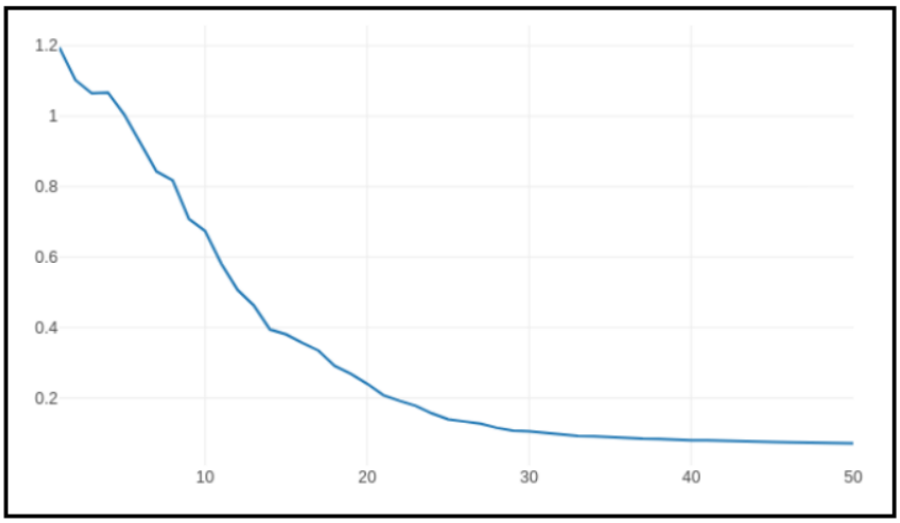


FIGURE 2.12 – La fonction de perte diminue avec le nombre d'époques[2]

6 Optimiseurs

La partie la plus importante de la formation du modèle est l'optimiseur. Imaginons la structure d'un modèle. La structure crée en définissant la séquence de couches avec le nombre de neurones, les fonctions d'activation et la forme d'entrée et de sortie est initialisée avec des poids aléatoires au début. Les poids qui ont déterminé l'influence d'un neurone sur le neurone suivant ou la sortie finale sont mis à jour par le réseau au cours du processus d'apprentissage. En bref, un réseau avec des poids aléatoires et une structure définie est le point de départ d'un modèle. Le réseau prend un échantillon d'entraînement et utilise ses valeurs comme entrées pour les neurones de la première couche, qui produisent ensuite une sortie avec la fonction d'activation définie. La sortie devient alors une entrée pour la couche suivante, et ainsi de suite. La sortie de la dernière couche serait la prédiction pour l'échantillon d'entraînement. C'est là que la fonction de perte entre en jeu. La fonction de perte aide le réseau à comprendre dans quelle mesure le jeu de poids actuel s'est bien ou mal comporté sur l'échantillon d'entraînement. L'étape suivante pour le modèle est de réduire la perte. Comment le réseau sait-il quelles étapes ou quelles mises à jour il doit effectuer sur les poids pour réduire la perte ? L'optimiseur l'aide à comprendre cette étape. La fonction d'optimisation est un algorithme mathématique qui utilise les dérivées, les dérivées partielles et la règle de la chaîne dans le calcul pour comprendre l'ampleur des changements que le réseau observera dans la fonction de perte en apportant une petite modification au poids des neurones. La modification de la fonction de perte, qui serait une augmentation ou une diminution, aide à déterminer la direction du changement nécessaire dans le poids de la connexion.

Pour résumer, on peut dire que l'optimiseur utilise le score obtenu par la fonction de perte pour ajuster un peu la valeur des poids, dans un sens qui fera baisser le score de perte, comme le montre la figure 2.13. L'optimiseur met en œuvre ce que l'on appelle l'algorithme de rétropropagation : l'algorithme central de l'apprentissage profond [9].

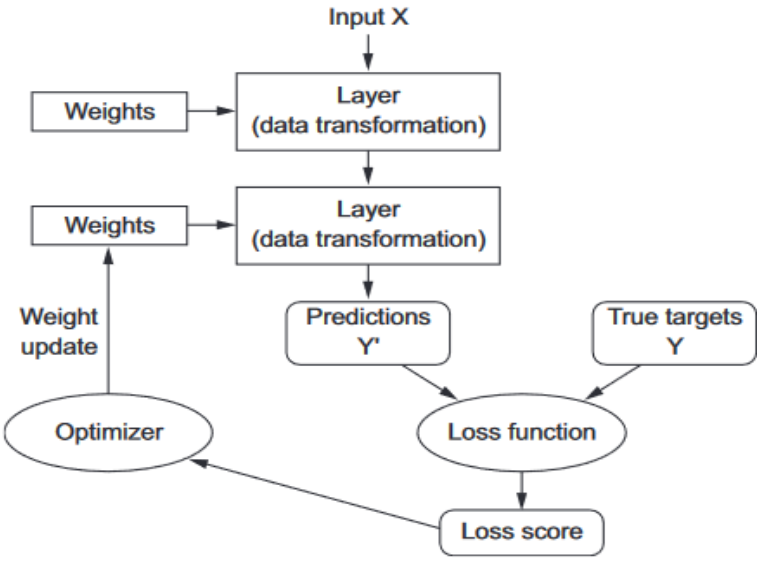


FIGURE 2.13 – Le score de perte est utilisé comme un signal de retour pour ajuster les poids.[9]

6.1 Stochastic Gradient Descent (SGD)

SGD effectue une itération avec chaque échantillon d'entraînement (c'est-à-dire qu'après le passage de chaque échantillon d'entraînement, il calcule la perte et met à jour le poids). Comme les poids sont mis à jour trop fréquemment, la courbe de perte globale serait très bruyante. La formule de mise à jour des poids peut être exprimée de manière simple, comme suit :

$$Poids = Poids - tauxd'apprentissage * Perte$$

Lorsque le taux d'apprentissage est un paramètre que nous définissons dans le l'architecture de réseau. Disons, pour un taux d'apprentissage =0,01.

Pour les mises à jour avec chaque échantillon de formation, il faudrait utiliser batch_size=1 dans la fonction de formation du modèle.

Pour réduire les fortes fluctuations dans les optimisations des SGD, une meilleure approche consisterait à réduire le nombre d'itérations en fournissant un mini lot, ce qui permettrait ensuite de faire la moyenne des pertes pour tous les échantillons d'un lot et de mettre à jour les poids à la fin du lot. Cette approche a eu plus de succès et permet un processus d'entraînement plus fluide. La taille des lots est généralement fixée en puissances de 2 (c'est-à-dire 32, 64, 128, etc.).

6.2 Momentum

Imaginons une boule de bowling roulant en pente douce sur une surface lisse : elle démarrera lentement, mais elle prendra rapidement de l'élan jusqu'à ce qu'elle atteigne finalement sa vitesse terminale (s'il y a une certaine friction ou résistance de l'air). C'est l'idée très simple qui sous-tend l'optimisation Momentum, proposée par Boris Polyak en 1964[17]. En revanche, la Descente de Gradient régulière ne fera que de petits pas réguliers sur la pente, de sorte qu'il faudra beaucoup plus de temps pour atteindre le fond. Rappelons que la Descente de Gradient met simplement à jour les poids Θ en soustrayant directement le gradient de la fonction de coût J(Θ) en ce qui concerne les poids (∇_θJ(Θ)) multiplié par le taux d'apprentissage η. L'équation est la suivante : $\theta \longleftarrow \theta - \eta \nabla_\theta J(\theta)$. Elle ne se soucie pas de savoir quels étaient les gradients antérieurs. Si le gradient local est minuscule, il va très lentement. L'optimisation du momentum se soucie beaucoup de ce qu'étaient les gradients précédents : à chaque itération, elle ajoute le gradient local au vecteur momentum m (multiplié par le taux d'apprentissage η), et elle met à jour les poids en soustrayant simplement ce vecteur de momentum. En d'autres termes, le gradient est utilisé comme une accélération, et non comme une vitesse. Pour simuler une sorte de mécanisme de friction et empêcher que l'impulsion ne devienne trop importante, l'algorithme introduit un nouvel hyperparamètre β, simplement appelé impulsion, qui doit être réglé entre 0 (friction élevée) et 1 (pas de friction). Une valeur d'impulsion typique est de 0,9, l'algorithme Momentum :

$$1 \,.\, \mathrm{m} \longleftarrow \beta \mathrm{m} + \eta \nabla_\theta \mathrm{J}(\theta)$$
$$2 \,.\, \theta \longleftarrow \theta - m$$

6.3 Nesterov Accelerated Gradient (NAG)

Une petite variante de l'optimisation Momentum, proposée par Yurii Nesterov en 1983[18], est presque toujours plus rapide que l'optimisation vanilla Momentum. L'idée de l'optimisation Momentum Nesterov, ou Nesterov Accelerated Gradient (NAG), est de mesurer le gradient de la fonction de coût non pas à la position locale mais légèrement en avant dans la direction du momentum. La seule différence avec l'optimisation vanilla Momentum est que le gradient est mesuré à $\theta + \beta m$ plutôt qu'à θ, l'algorithme Nesterov Accelerated Gradient :

$$1 \,.\, \mathrm{m} \longleftarrow \beta \mathrm{m} + \eta \nabla_\theta \mathrm{J}(\theta + \beta m)$$
$$2 \,.\, \theta \longleftarrow \theta - m$$

Ce petit ajustement fonctionne parce qu'en général, le vecteur d'impulsion pointe dans la bonne direction (c'est-à-dire vers l'optimum), il est donc légèrement plus précis d'utiliser le gradient mesuré un peu plus loin dans cette direction plutôt que d'utiliser le gradient à la position initiale. Ce petit ajustement fonctionne parce qu'en général, le vecteur d'impulsion pointe dans la bonne direction (c'est-à-dire vers l'optimum), il est donc légèrement plus précis d'utiliser le gradient mesuré un peu plus loin dans cette direction plutôt que d'utiliser le gradient à la position initiale, comme nous pouvons le voir sur la figure 18 (où ∇_1 représente le gradient de la fonction de coût mesurée au point de départ θ, et ∇_2 représente le gradient au point situé à $\theta + \beta m$). Comme nous pouvons le voir, la mise à jour de Nesterov se rapproche légèrement de l'optimum. Au bout d'un certain temps, ces petites améliorations s'additionnent et NAG finit par être nettement plus rapide que l'optimisation régulière de Momentum. De plus, noter que lorsque momentum pousse les poids à travers une vallée, ∇_1 continue à pousser plus loin à travers la vallée, tandis que ∇_2 repousse vers le fond de la vallée. Cela permet de réduire les oscillations et donc de converger plus rapidement. La figure 2.14 montre la différence entre Momentum classique et *Nesterov Accelerated Gradient* .

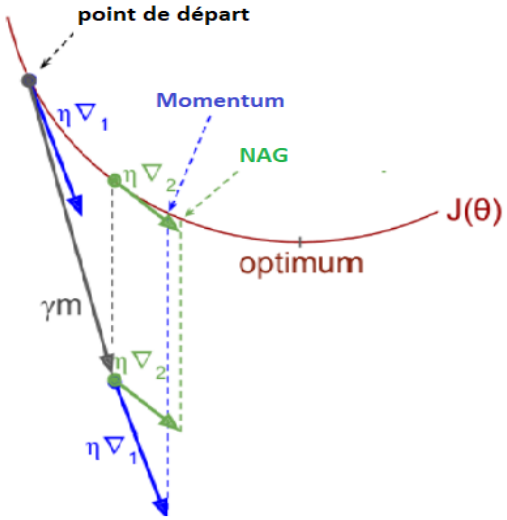


FIGURE 2.14 – Momentum vs Nesterov Accelerated Gradient (NAG)[10]

6.4 AdaGrad

Imaginons le problème du bol allongé : La descente commence en descendant rapidement la pente la plus raide, puis descend lentement le fond de la vallée. Il serait bon que l'algorithme puisse détecter cela très tôt et corriger sa direction pour pointer un peu plus vers l'optimum global.
L'algorithme AdaGrad [19] réalise cela en réduisant le vecteur de gradient le long des dimensions les plus importantes.

En bref, cet algorithme diminue le taux d'apprentissage, mais il le fait plus rapidement pour les dimensions à forte pente que pour les dimensions à faible pente. Il s'agit d'un taux d'apprentissage adaptatif, qui permet d'orienter les mises à jour résultantes plus directement vers l'optimum global. Un avantage supplémentaire est qu'il nécessite beaucoup moins de réglage de l'hyperparamètre du taux d'apprentissage η.

AdaGrad fonctionne souvent bien pour les problèmes quadratiques simples, mais malheureusement il s'arrête souvent trop tôt lors de l'apprentissage des réseaux de neurones. Le taux d'apprentissage est tellement réduit que l'algorithme finit par s'arrêter complètement avant d'atteindre l'optimum global. nous ne devons donc pas l'utiliser pour entraîner des réseaux de neurones profonds (il peut cependant être efficace pour des tâches plus simples telles que la régression linéaire).

6.5 RMSProp

Bien qu'AdaGrad ralentisse un peu trop vite et finisse par ne jamais converger vers l'optimum global, l'algorithme RMSProp[20] corrige ce problème en accumulant uniquement les gradients des itérations les plus récentes (par opposition à tous les gradients depuis le début de l'entraînement). Pour ce faire, il utilise la décroissance exponentielle dans la première étape, Algorithme RMSProp :

$$1 \text{ . s} \longleftarrow \beta s + 1 - \beta \nabla_\theta J\theta \otimes \nabla_\theta J\theta$$
$$2 \text{ . } \theta \longleftarrow \theta - \eta \nabla_\theta J\theta \oslash s + \varepsilon$$

Le taux de décroissance β est généralement fixé à 0,9. Il s'agit d'un hyperparamètre, mais cette valeur par défaut fonctionne souvent bien, de sorte que nous n'avons peut-être pas besoin de l'ajuster du tout. Sauf pour des problèmes très simples, cet optimiseur est presque toujours plus performant qu'AdaGrad. Il est aussi généralement plus performant que l'optimisation Momentum et Nesterov Accelerated Gradients. En fait, c'était l'algorithme d'optimisation préféré de nombreux chercheurs jusqu'à ce que l'optimisation d'Adam soit mise en place.

6.6 Adam

Adam[21] , qui signifie « *Adaptive Moment Estimation* », est de loin l'optimiseur le plus populaire et le plus utilisé dans le domaine de l'apprentissage profond. Dans la plupart des cas, nous pouvons choisir l'optimiseur Adam à l'aveuglette et oublier les autres possibilités d'optimisation. Cette technique d'optimisation calcule un taux d'apprentissage adaptatif pour chaque paramètre. Cette technique d'optimisation calcule un taux d'apprentissage adaptatif pour chaque paramètre. Elle définit la dynamique et la variance du gradient de la perte et exploite un effet combiné pour mettre à jour les paramètres de poids. La représentation mathématique peut être simplifiée de la manière suivante :

$$poids = poids - \text{(Momentum et variance combinés)}$$

Adam combine les idées de l'optimisation de la dynamique et de RMSProp : tout comme l'optimisation de la dynamique, elle suit une moyenne des gradients passés qui décroît de manière exponentielle, et tout comme RMSProp, elle suit une moyenne des gradients carrés passés qui décroît de manière exponentielle.

En fait, comme Adam est un algorithme de taux d'apprentissage adaptatif (comme AdaGrad et RMSProp), il nécessite moins de réglage de l'hyperparamètre de taux d'apprentissage η. Nous pouvons souvent utiliser la valeur par défaut $\eta = 0{,}001$, ce qui rend Adam encore plus facile à utiliser que l'algorithme du gradient.

7 Conclusion

Dans ce second chapitre, nous nous sommes intéressés au domaine de l'apprentissage par réseaux de neurones, tout en faisant le tour de ses différents points sous jacent. En effet, nous avons vue le passage des neurones biologiques aux neurones artificiels, les différentes couches de ses réseaux, ce qu'est une fonction d'activation, tout en définissant quelques unes d'entre elles. Par ailleurs, nous avons également décrie ce qu'est une fonction de perte. Pour finir, nous avons vue ce qu'est un optimiseur, ainsi que ses différents types.

Le domaine de l'estimation immobilière est un domaine en constante recherche, le but étant de réaliser des modèles d'apprentissage automatique ayant une faible marge d'erreur. En effet, le chapitre suivant sera consacré à la présentation de certains modèles que nous allons entrainer et tester par la suite.

Chapitre 3

Modèles proposés

1 Introduction

L'un des défis majeurs de l'estimation immobilière consiste à trouver des modèles d'apprentissage automatique pouvant prédire l'estimation qui se rapproche le plus de la valeur du bien immobilier. En effet, pour arriver à cela il est primordial de préparer les données d'entrées de ces modèles car les modèles d'apprentissage automatique s'entrainent sur des collections de données. Ces collections de données comprennent beaucoup de problèmes tels que : des données manquantes, attributs catégoriques et textuelles, attributs numériques à échelles différentes, etc.

Plusieurs techniques de prétraitement et de modèles d'apprentissage automatique ont été proposés pour résoudre des problématiques de régressions. Cependant, en ce qui concerne l'estimation immobilière, la question reste en suspend pour trouver les techniques de prétraitement adaptées pour les collections de données immobilières ainsi que les modèles ayant une marge d'erreur la plus optimale.

Ainsi dans ce chapitre nous abordons les différentes étapes et techniques de prétraitement à effectuer sur les données d'entrée et nous proposons trois modèles d'apprentissage automatique réalisant l'estimation de la valeur marchande de propriétés immobilières.

Ce chapitre est organisé comme suit. Nous présentons les motivations et les objectifs que nous nous sommes fixés. Ensuite, nous présentons un état de l'art sur ce qui se fait pour l'estimation immobilière. Dans la troisième section, nous donnons les différentes techniques de prétraitement utilisées. Par la suite nous présentons les trois modèles proposées pour l'estimation de la valeur marchande de propriétés immobilières. Enfin, nous terminons par une conclusion.

2 Motivation et objectifs

L'objectif principal de notre travail est de construire des modèles d'apprentissage automatique adaptés pour l'estimation de la valeur de propriétés immobilières ayant une faible marge d'erreur. Avant la réalisation de ces modèles nous devons tout d'abord effectuer des prétraitements sur la collection de données immobilières afin qu'elles puissent être exploitées par les modèles pour leurs entrainement et leurs tests. Par la suite, nous allons construire trois types de modèles différents réalisant l'estimation des biens immobiliers. Enfin, nous effectuons une comparaison des résultats de chacun des modèles.

3 Etat de l'art de l'estimation immobilière

L'apprentissage automatique a été utilisé dans des disciplines tel que le commerce, l'ingénierie informatique, l'ingénierie industrielle, la bio-informatique, la médecine, la pharmacie, la physique et les statistiques pour rassembler des connaissances et prédire des événements futurs.
Ces dernières années, en raison de la tendance croissante aux « *Big Data* », l'apprentissage automatique est devenu une approche de prédiction essentielle, car il permet de prévoir les prix de l'immobilier avec plus de précision en fonction de leurs caractéristiques. Plusieurs études ont exploré ce problème et ont prouvé la pertinence de l'approche de l'apprentissage automatique.
Pour trouver des recherches sur le prix du logement, il faut surtout consulter les revues économiques,

financières et immobilières. Un certain nombre de chercheurs ont tenté de décrire les variables affectant la dynamique ou les mouvements des prix de l'immobilier. Pour étudier comment les informations sur les conditions économiques locales affectent le prix du logement, Favara et Song[22] décrivent un modèle mathématique de coût d'utilisation basé sur l'hypothèse que l'offre de logement est inélastique et que les variations de la demande sont stochastiques. Borowiecki[23] note que les coûts de construction et la croissance démographique sont parmi les facteurs importants qui influencent le prix des logements en Suisse, tandis que Das et al[24] présentent un modèle autorégressif bayésien (Spackovaand Straub[25] ; Huang et al[26]) pour prédire le taux de croissance annualisé du prix de l'immobilier pour le marché du logement de petite taille en Afrique du Sud en tenant compte des facteurs suivants : revenus, taux d'intérêt, coûts de construction, variables du marché du travail, cours des actions, production industrielle, indice de confiance des consommateurs et facteurs représentant l'économie mondiale. Égert et Mihaljek[27] indiquent que le produit intérieur brut (PIB) par habitant et les taux d'intérêt ont la plus grande influence sur le logement.

Plusieurs entreprises d'annonce immobilière tel que Zillow utilise l'apprentissage automatique afin de pouvoir donner une estimations des biens en ventes sur leur site. Malgré l'intérêt porté par la communauté scientifique à ce domaine, beaucoup de méthodes restent à explorer afin de réaliser des modèles d'apprentissage automatique ayant une précision optimale. Dans ce mémoire, le but est de proposer quelques modèles d'apprentissage automatique pour prédire le prix des logements et de comparer les résultats de ces derniers.

4 Prétraitement

Afin de construire un modèle d'apprentissage automatique, il nous faut un ensemble de données, en l'occurrence une collection de données, qui en générale comporte des données manquantes, des attributs catégoriques, des données à échelles différentes, des formats de données différents, etc...
C'est pour cela qu'il est nécessaire d'effectuer un ensemble d'étapes dans le but de préparer la collection de données afin d'entrainer puis de tester un modèle d'apprentissage automatique.

4.1 Nettoyage des données

La plupart des algorithmes d'apprentissage automatique ne peuvent pas fonctionner avec les paramètres manquants. Si un attribut à des valeurs manquantes, alors pour corriger cela nous avons trois options :

- Option 1 : Supprimer les lignes qui ne contiennent pas de valeurs pour cet attribut.
- Option 2 : Supprimer l'attribut.
- Option 3 : Fixer les valeurs à une certaine valeur (zéro, la moyenne, la médiane, etc.).

Si on choisit l'option 3, on doit calculer la valeur médiane de l'ensemble de formation, et l'utiliser pour remplir les valeurs manquantes de l'ensemble de formation, mais n'oublions pas non plus de sauvegarder la valeur médiane calculée afin de l'utilisé plus tard pour remplacer les valeurs manquantes dans l'ensemble de test lorsqu'on voudra évaluer le système, et aussi une fois que le système sera opérationnel pour remplacer les valeurs manquantes dans les nouvelles données.

4.2 Formats d'encodage des attributs catégoriques

Les modèles d'apprentissage profond ne comprennent que des données numériques. Par conséquent, toutes les caractéristiques catégoriques stockées sous forme de colonnes de texte doivent être converties en une forme codée one-hot ou binary pour les données d'entraînement du modèle.

4.2.1 One-hot

L'encodage one-hot est un processus simple de représentation d'une colonne de catégories sous forme de matrice binaire élargie étiquetée. Ainsi, une caractéristique catégorique avec trois valeurs distinctes, par exemple "Classe A", "Classe B" et "Classe C", peut être représentée par trois colonnes au lieu d'une, où chaque colonne représenterait un drapeau binaire pour une valeur de catégorie individuelle. L'exemple suivant « figure 3.1 » résume cette situation.
Le plus grand problème avec l'encodage one-hot est qu'il nécessite un grand nombre de nouvelles variables, généralement on préfère utiliser l'encodage one-hot lorsqu'on a quelques catégories.

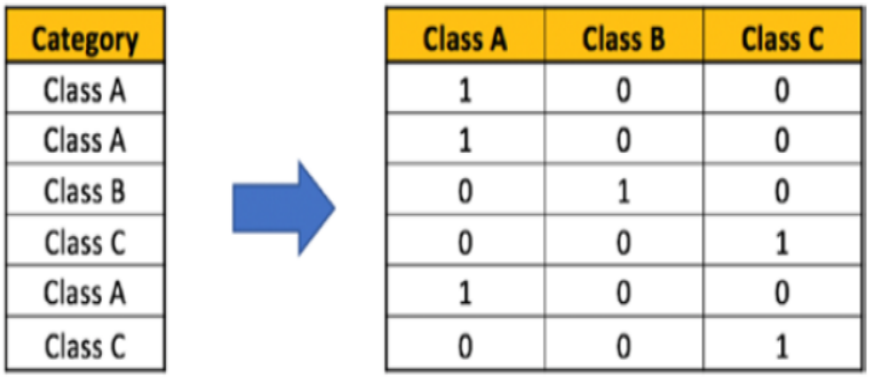


FIGURE 3.1 – Encodage one-hot [6]

4.2.2 Binary

Cette technique n'est pas aussi intuitive que la précédente. Dans cette technique, les catégories sont d'abord codées en ordinal, puis ces entiers sont convertis en code binaire, puis les chiffres de cette chaîne binaire sont divisés en colonnes séparées. Les données sont ainsi codées en moins de dimensions qu'en one-hot.
Le problème avec l'encodage binary est qu'il introduit certaines corrélations injustifiées entre les catégories, généralement on préfère utiliser l'encodage binary lorsqu'on a beaucoup de catégories.

4.3 Mise à l'échelle « *Feature Scaling* »

L'une des transformations les plus importantes à appliquer aux données est la mise à l'échelle. À quelques exceptions près, les algorithmes d'apprentissage automatique ne fonctionnent pas bien lorsque les attributs numériques d'entrée ont des échelles très différentes. C'est le cas des données sur le logement. Il existe deux moyens communs de faire en sorte que tous les attributs aient la même échelle : normalisation et la standardisation.

4.3.1 Normalisation

La normalisation (ou Min-max scaling) est assez simple : les valeurs sont réajustées et redimensionnées de sorte qu'elles finissent par aller de 0 à 1. Pour se faire, nous soustrayons la valeur min et divisons par le max moins le min. Scikit-Learn fournit pour cela un transformateur appelé MinMaxScaler. Il dispose d'un hyperparamètre feature_range qui permet de modifier la plage si nous ne souhaitons pas obtenir 0-1 pour une raison quelconque.

4.3.2 Standardisation

La standardisation est très différente : elle soustrait d'abord la valeur moyenne (les valeurs standardisées ont donc toujours une moyenne nulle), puis elle divise par la variance, de sorte que la distribution résultante présente une variance unitaire. Contrairement à la mise à l'échelle min-max, la standardisation ne limite pas les valeurs à une plage spécifique, ce qui peut poser un problème pour certains algorithmes (par exemple, les réseaux de neurones attendent souvent une valeur d'entrée allant de 0 à 1). Cependant, la standardisation est beaucoup moins affectée par les valeurs aberrantes. Par exemple, supposons qu'un district ait un revenu médian égal à 100 (par erreur). L'échelle min-max écraserait alors toutes les autres valeurs de 0-15 à 0-0,15, alors que la standardisation ne serait pas beaucoup affectée.

5 Modèles implémentés

Dans ce qui suit, nous décrivons les modèles proposés pour l'estimation de la valeur de propriétés immobilières. En effet, dans ce mémoire, nous proposons trois types de modèles.

le modèle 1 utilise la technique d'apprentissage ensembliste *Boosting*. Le modèle 2 s'appuie compte à lui sur une autre technique d'apprentissage ensembliste *Bagging*. Pour le modèle 3, nous optons pour la création d'un modèle à base de neurone utilisant une architechture ANN classique.

5.1 Modèle 1

Pour ce premier modèle, nous allons utiliser un algorithme venant de La bibliothèque XGBoost qui implémente l'algorithme de l'arbre de décision « *gradient boosting* ». Le boosting est une technique ensembliste où de nouveaux modèles sont ajoutés pour corriger les erreurs commises par les modèles existants. Les modèles sont ajoutés de manière séquentielle jusqu'à ce qu'aucune amélioration ne soit plus possible. Un exemple populaire est le AdaBoost algorithm qui pondère les points de données difficiles à prévoir. Le gradient boosting est une approche dans laquelle de nouveaux modèles sont créés pour prédire les résidus ou les erreurs des modèles précédents et sont ensuite ajoutés pour faire la prédiction finale. Elle est appelée « *gradientboosting* » parce qu'elle utilise un algorithme de descente de gradient pour minimiser la perte lors de l'ajout de nouveaux modèles. La descente du gradient est une technique itérative qui permet d'approcher la solution d'un problème d'optimisation. En apprentissage supervisé, la construction du modèle revient souvent à déterminer les paramètres (du modèle) qui permettent d'optimiser (max ou min) une fonction objectif. Cette approche permet de résoudre les problèmes de modélisation prédictive de régression et de classification. Pour notre cas nous allons utiliser l'algorithme XGBoost adapté pour la régression « XGBRegressor ».

5.2 Modèle 2

Pour ce second modèle, nous allons utiliser la technique d'apprentissage ensembliste *Bagging* et cela en utilisant l'algorithme des forêts aléatoire appliquer à la régression et cela en créant plusieurs arbres de décisions et d'entrainer chaque arbre sur une portion alétoire de notre collection de données, après avoir entrainé chaque arbre, nous pouvons ensuite regrouper les résultats de chacun des arbres afin de faire la prédiction finale.

5.3 Modèle 3

Pour ce modèle on crée une architecture ANN classique décrite dans la figure « figure 3.2 », on crée un modèle fonctionnel avec une couche d'entée des paramètres, 6 couches cachées denses disposons de la fonction d'activation 'Relu' et d'une couche de sortie qui est la prédiction, une fonction objectif 'mean_absolute_error' et l'optimiseur 'Adam' pour entrainer le modèle.

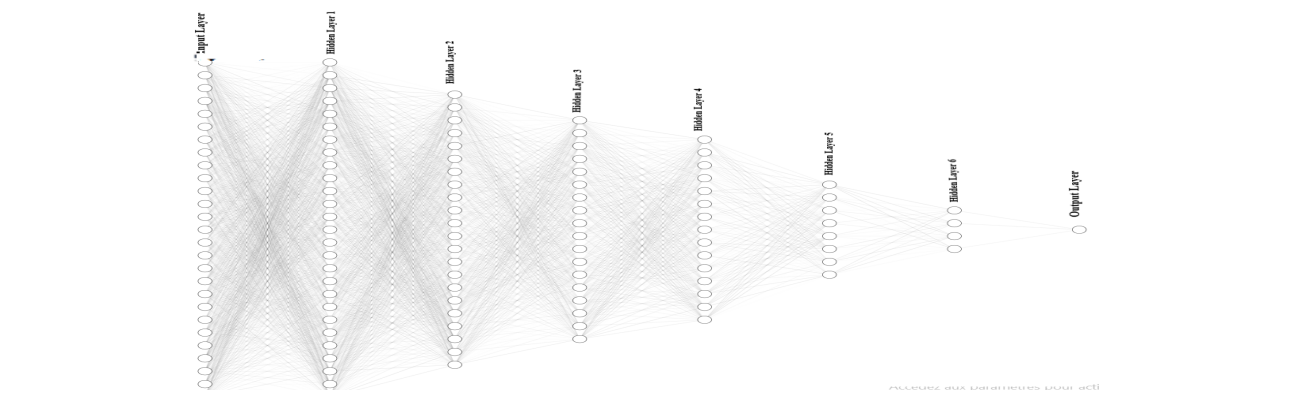


FIGURE 3.2 – ANN classique

6 Conclusion

Dans ce troisième chapitre, nous avons tout d'abord vus un état de l'art sur ce qui se fait en matière d'estimation immobilière. Ensuite, nous avons présenté les différentes phases et techniques de prétraitement utilisées pour préparer les données d'entré aux modèles d'apprentissage automatique ainsi que les avantages et les inconvénients de ses méthodes. Enfin, nous avons décrit les trois modèles proposés pour l'estimation de la valeur de propriétés immobilières.

Le dernier chapitre est consacré à l'implémentation et à l'évaluation de ces trois modèles en utilisant certaines techniques de prétraitement vue dans ce chapitre.

Chapitre 4

Implémentation et évaluation des modèles proposés

1 Introduction

Dans ce chapitre nous présentons l'implémentation et l'évaluation de trois modèles d'apprentissage automatique proposés et décris dans le chapitre précèdent.

L'implémentation consiste à créer les modèles qui sont entrainés puis tester sur une collection de données, afin de pouvoir comparer les performances de ces derniers. La collection passe tout d'abord par une phase de prétraitement en utilisant les techniques décrites dans le chapitre précèdent.

Afin de pouvoir réaliser le code prévu pour l'implémentation, il nous faut connaitre les différentes librairies utilisées dans le domaine de l'apprentissage automatique pour réaliser notamment les prétraitements nécessaires sur la collection de données. Ensuite, nous implémentons les trois modèles en effectuant leurs entrainements puis leurs tests. À cet effet, nous avons obtenus des résultats concluants, en particulier pour le modèle 3.

Dans ce chapitre, nous allons tout d'abord décrire les différentes bibliothèques utilisées ainsi que leurs utilitées. Par la suite, nous décrivons les prétraitements réalisés sur la collection de données ainsi que l'implémentation des modèles décrie. Enfin, avant de conclure ce chapitre nous effectuons une évaluation des résultats obtenus.

2 Bibliothèques utilisées

2.1 Numpy

NumPy est l'abréviation de « *Numerical Python* » et c'est un ensemble fondamental pour le calcul scientifique en Python. NumPy fournit à Python une vaste bibliothèque mathématique capable d'effectuer des calculs numériques de manière efficace et efficiente afin de pouvoir travailler avec des tableaux multidimensionnels et des structures de données matricielles, très courante dans les domaines de la science des données et de l'apprentissage automatique.
Pourquoi utiliser NumPy ?

- Les tableaux Numpy utilisent moins de mémoire que les listes python normales.
- Une liste python normale est un groupe de pointeurs pour séparer les objets Python (par exemple les nombres à l'intérieur de la liste).
- Un tableau numpy est conçu pour être un tableau de valeurs uniformes, sans utiliser d'espace pour les pointeurs de type.
- Numpy peut également lire les informations plus rapidement et dispose de nombreuses opérations de diffusion pratiques qui peuvent être effectuées à travers les dimensions du tableau.

2.2 Pandas

Pandas est un package pour la manipulation et l'analyse de données en Python. Le nom Pandas est dérivé du terme « *Panel Data* ». Pandas intègre deux structures de données supplémentaires dans Python, à savoir Pandas Series et Pandas DataFrame. Ces structures de données nous permettent de travailler avec des données étiquetées et relationnelles de manière simple et intuitive.

Le succès récent des algorithmes d'apprentissage automatique est en partie dû aux énormes quantités de données dont nous disposons pour former nos algorithmes. Cependant, lorsqu'il s'agit de données, la quantité n'est pas la seule chose qui compte, la qualité des données est tout aussi importante. Il arrive souvent que de grands ensembles de données ne soient pas prêts à être intégrés dans les algorithmes d'apprentissage. Le plus souvent, les grands ensembles de données auront souvent des valeurs manquantes, des valeurs aberrantes, des valeurs incorrectes, etc... Avoir des données avec beaucoup de valeurs manquantes ou mauvaises, par exemple, ne va pas permettre aux algorithmes d'apprentissage automatique de bien fonctionner. Par conséquent, une étape très importante de l'apprentissage automatique consiste à examiner d'abord les données et à s'assurer qu'elles sont bien adaptées à votre algorithme de formation en effectuant une analyse de base des données. C'est là que les pandas entrent en jeu. Les Pandas Series et les DataFrames sont conçus pour une analyse et une manipulation rapides des données, tout en étant flexibles et faciles à utiliser. Voici quelques caractéristiques qui font des Pandas un excellent ensemble pour l'analyse des données :

- Permet l'utilisation d'étiquettes pour les lignes et les colonnes.
- Peut calculer des statistiques continues sur des données de séries chronologiques.
- Manipulation aisée des valeurs NaN.
- Est capable de charger des données de différents formats dans des DataFrames.
- Peut réunir et fusionner différents ensembles de données.
- Il s'intègre avec NumPy et Matplotlib.

Pour ces raisons, entre autres, les Pandas DataFrames sont devenus l'un des objets Pandas les plus utilisés pour l'analyse des données en Python.

2.3 Matplotlib

Ce paquet est la bibliothèque Python la plus populaire actuellement pour produire des tracés et autres visualisations de données en 2D. Comme l'analyse des données nécessite des outils de visualisation, c'est la bibliothèque qui convient le mieux à cette fin.

On peut facilement générer des graphiques, des histogrammes, des diagrammes à barres, des diagrammes de dispersion et bien d'autres en utilisant Python et quelques lignes de code. Au lieu de passer du temps à chercher des solutions, nous pouvons nous concentrer sur la génération de diagrammes pour une analyse et une exploration plus rapides des données.

Parmi toutes les caractéristiques qui en ont fait l'outil le plus utilisé dans la représentation graphique des données, il y en a quelques-unes qui ressortent :

- Développement progressif et visualisation interactive des données.
- Un contrôle plus strict des éléments graphiques.
- Exporter vers de nombreux formats, tels que PNG, PDF, SVG et EPS.

Les fonctionnalités de traçage intégrées de Seaborn et de Pandas sont toutes deux construites à l'aide de matplotlib.

2.4 Seaborn

Seaborn est une bibliothèque de visualisation construite à l'aide de matplotlib qui se concentre sur la création de tracés statistiques standard avec un simple appel de fonction d'une ligne. Cela se fait au prix de moins d'options de personnalisation que le pur matplotlib.

Cependant, on peut toujours modifier les attributs des tracés créés dans seaborn en ajoutant des appels de fonctions dans matplotlib.pyplot.

2.5 Scikit-Learn

Scikit-Learn est une librairie d'apprentissage automatique construite à partir de NumP, SciPy et Matplotlib. Scikit-Learn offre des outils simple et efficase pour les taches commune en analyse de données, telle que la classification, la régression, le clusturing, réduction de la dimensionnalité, et le prétraitement.

2.6 Tensorflow

TensorFlow est une interface de programmation scalable et multi-plateforme pour la mise en œuvre et l'exécution d'algorithmes d'apprentissage automatique.
TensorFlow a été développé par les chercheurs et les ingénieurs de l'équipe Google Brain. TensorFlow a été initialement conçu pour un usage interne à Google, mais il a ensuite été publié en novembre 2015 sous une licence open source.

Afin d'améliorer les performances des modèles d'apprentissage automatique, TensorFlow permet une exécution à la fois sur les CPU et les GPU. Toutefois, ses plus grandes capacités de performance peuvent être découvertes lors de l'utilisation des GPU.

TensorFlow prend actuellement en charge les interfaces frontend pour un certain nombre de langages de programmation. L'API Python de TensorFlow est actuellement l'API la plus complète, attirant ainsi de nombreux praticiens de l'apprentissage automatique et de l'apprentissage profond. En outre, TensorFlow possède une API officielle en C++.
TensorFlow possède un vaste écosystème de composants connexes, notamment des bibliothèques comme Tensorboard, des API de déploiement et de production.

TensorFlow 1.x avait un système complexe de classes python pour la construction de modèles, et en raison de l'énorme popularité de Keras, lorsque TensorFlow 2.0 est sorti, TF a adopté Keras comme API officielle pour TensorFlow.

2.7 Keras

Keras est une API d'apprentissage profond de haut niveau qui rend très simple la formation et l'exploitation de réseaux de neurones. Elle peut fonctionner sur TensorFlow, Theano ou Microsoft Cognitive Toolkit (anciennement connu sous le nom de CNTK). TensorFlow est livré avec sa propre implémentation de cette API, appelée tf.keras, qui prend en charge certaines fonctionnalités avancées de TensorFlow (par exemple, pour charger efficacement des données).

Bien que Keras reste une bibliothèque distincte de Tensorflow, elle peut désormais être officiellement importée par TensorFlow, il n'est donc pas nécessaire de l'installer.

L'API de Keras est facile à utiliser et permet de construire des modèles en ajoutant simplement des couches les unes sur les autres par de simples appels.

3 Prétraitement de la collection de données

Afin d'Implémentation et évaluer un modèle d'apprentissage automatique pour l'estimation de la valeur marchande de propriétés immobilières nous utiliserons la collection de données proposé par le site Zillow qui est une entreprise d'annonce immobilière, la collection est héberger sur Kaggle via le lien suivant (https ://www.kaggle.com/c/zillow-prize-1/data).

Cette collection contient énormément de données manquantes, comme le montre la figure 4.1. Par conséquent nous allons devoir la nettoyer avec les différentes techniques que nous avons décrites dans le chapitre 3, beaucoup de paramètres ont dû donc être supprimer. La collection contient également des attributs catégoriques, pour cela nous allons utiliser le format one-hot pour convertir ses attributs :

- propertycountylandusecode : Nous obtenons 78 nouveaux attributs (propertycountylandusecode_0, ...) ;
- buildingqualitytypeid : Nous obtenons 6 nouveaux attributs (buildingqualitytypeid_0, ...) ;
- buildingclasstypeid : Nous obtenons 9 nouveaux attributs (buildingclasstypeid_0, ...) ;
- propertylandusetypeid : Nous obtenons 6 nouveaux attributs (propertylandusetypeid_0, ...) ;
- regionidcity : Nous obtenons 120 nouveaux attributs (regionidcity_0, ...) ;
- regionidzip : Nous obtenons 271 nouveaux attributs (regionidzip_0, ...) ;
- propertyzoningdesc : Nous obtenons 2097 nouveaux attributs (propertyzoningdesc_0, ...).

nous allons également créer 3 nouveaux paramètres dans notre dataset qui sont :

- count_landusecode : Le nombre de maisons ; disponible dans le comté ou se trouve la maison ;
- count_city : Le nombre de maisons disponible dans la vile ou se trouve la maison ;

- count_zip : Le nombre de maisons disponible ayant le même code postal que la maison.

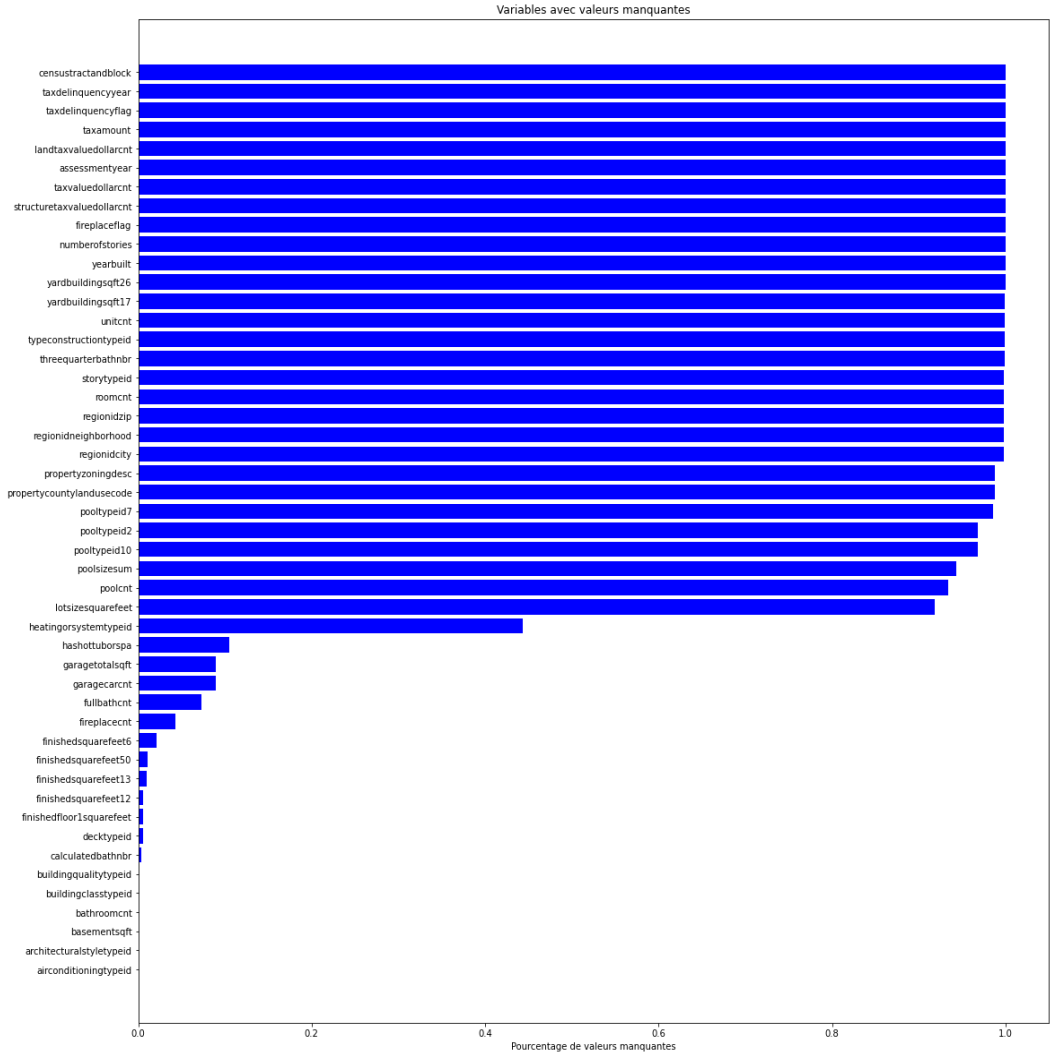


FIGURE 4.1 – Pourcentage des données manquantes

par la suite nous allons diviser la collection de données en 2 parties (80 % pour la collection d'entrainement, 20 % pour la collection de test) et cela en utilisant la fonction proposée par Scikit-Learn sklearn.model_selection.train_test_split qui nous donne

- X_train : paramètres des propriétés immobilères de la collection d'entrainement.
- X_test : paramètres des propriétés immobilières de la collection de test.
- y_train : valeurs des propriétés immobilières de la collection d'entrainement.
- y_test : valeurs des propriétés immobilières de la collection de test.

Apres avoir diviser la collection de données, vient l'étape de mise à l'échelle qui se fera via la normalisation des paramètres des propriétés immobilières des collections d'entrainement et de test grâce à la fonction sklearn.preprocessing.MinMaxScaler, les paramètres concernés sont : bathroomcnt, bedroomcnt, calculatedfinishedsquarefeet, fullbathcnt, lotsizesquarefeet, poolcnt, rawcensustractandblock, unitcnt, yearbuilt, taxamount, count_landusecode.

4 Création des modèles et résultats

Dans ce qui suit nous allons créer les trois modèles, chacun d'eux sera entrainé puis tester sur les collections d'entrainement et de test respectivement.

Afin d'évaluer la précision des trois modèles, nous calculerons différentes mesures de performances après l'entrainement pour qu'à la fin nous puissions les comparer.

4.1 Modèle 1

Le modèle 1 est construit grâce à la fonction XGBRegressor de la bibliothèque XGBoost, les modèles venant de cette bibliothèques sont connus pour leurs rapidités, efficacités, flexibilités en terme de paramétrages des fonctions, la possibilité d'utiliser un *early_stopping* afin d'éviter le sur-apprentissage ainsi qu'une *evaluation_set* afin de permettre au modèle de savoir s'il améliore ses prédictions durant l'entrainement. La figure 4.2 montre les différents paramètres et valeurs utilisées par la fonction XGBRegressor afin de construire le modèle.

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, eval_metric='mae', gamma=0,
             importance_type='gain', learning_rate=0.05, max_delta_step=0,
             max_depth=3, min_child_weight=1, missing=None, n_estimators=1000,
             n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
             silent=None, subsample=1, verbosity=1)
```

FIGURE 4.2 – Configuration du modèle 1

Nous pouvons maintenant évaluer la précision du modèle sur la collection de test est cela en calculant des mesures de performances :

- **Root Mean Square Error (RMSE) :** Elle mesure l'écart-type des erreurs que le système commet dans ses prévisions. Par exemple, une RMSE égale à 50 000 signifie qu'environ 68 % des prévisions du système se situent à moins de 50 000 $ de la valeur réelle, et environ 95 % des prévisions se situent à moins de 100 000 $ de la valeur réelle.
- **Mean Absolute Error (MAE) :** MAE est la somme des différences absolues entre notre cible et la prévue. Il mesure donc l'ampleur moyenne des erreurs dans un ensemble de prédictions, sans tenir compte de leurs directions, ainsi un MAE de 20 000 indique qu'on moyenne le modèle est a 20 000 $ de la valeur réel.
- **Explained variance score :** Fonction du score de régression de la variance, Le meilleur score possible est de 1,0.

```
RMSE : 32118.61026882424
MAE : 14985.168100337934
EVS : 0.9036856819671103
```

FIGURE 4.3 – calcule des performances du modèle 1

Le graphe suivant « figure 4.4 » indique on rouge les valeurs attendues (prix des propriétés immobilières) et les points bleus sont les prédictions du modèle 1.

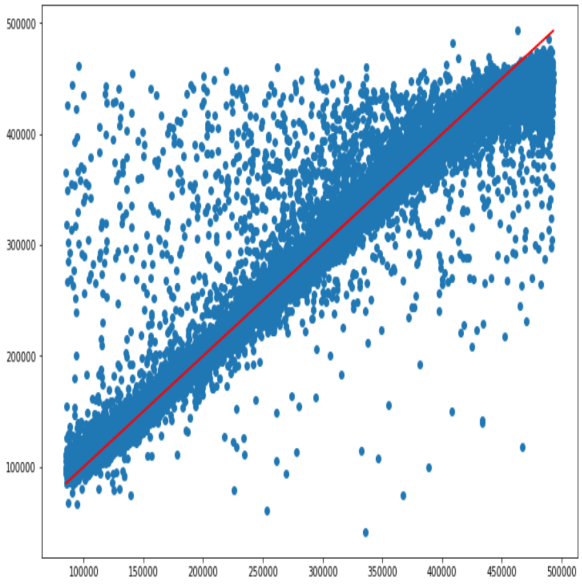


FIGURE 4.4 – valeurs attendues vs prédictions pour le modèle 1

4.2 Modèle 2

Le modèle 2 est construit selon la technique d'apprentissage ensembliste *Bagging* est cela en utilisant l'algorithme du Random Forest appliquer aux problèmes de régressions, qui est proposé par bibliothèque Scikit-Learn via la fonction « sklearn.ensemble.RandomForestRegressor() », malheureusement il n'y a pas la possibilité d'utiliser un *Early Stopping* vue que les arbres construits sont entraîner en Parallèle et il faut à l'avance préciser le nombres d'arbres que devra avoir la forêt aléatoire. La figure 4.5 montre les différents paramètres et valeurs utiliser par la fonction RandomForestRegressor() afin de construire la forêt aléatoire.

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mae',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=10, n_jobs=None, oob_score=False,
                      random_state=None, verbose=2, warm_start=False)
```

FIGURE 4.5 – Configuration du modèle 2

La figure 4.6 montre les résultats des calcules des mesures de performances sur la collection de test après l'entrainement du modèle.

```
RMSE : 32959.33603840718
MAE : 14608.068665367988
EVS : 0.898576783484554
```

FIGURE 4.6 – calcule des performances du modèle 2

Le graphe suivant « figure 4.7 » indique en rouge les valeurs attendues (prix des propriétés immobilières) et les points bleus sont les prédictions du modèle 2.

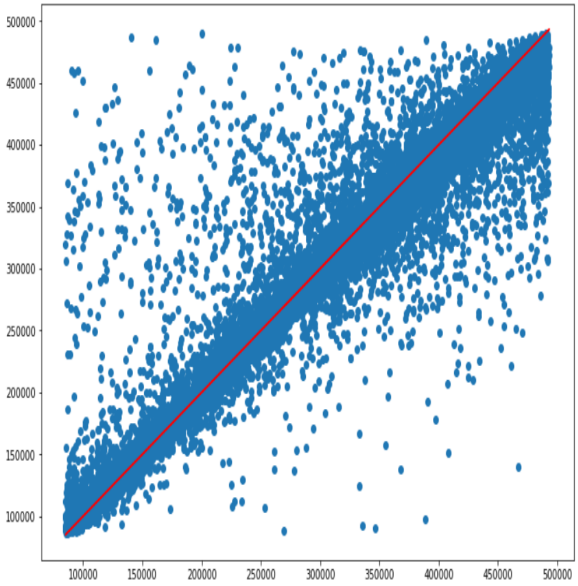


FIGURE 4.7 – valeurs attendues vs prédictions pour le modèle 2

4.3 Modèle 3

Comme décrit dans le chapitre 3, le modèle 3 est créé avec une architécture ANN classique, avec Keras on crée un modèle fonctionnel avec une couche d'entée de 507 paramètres, 6 couches cachées denses disposant de la fonction d'activation « Relu » et d'une couche de sortie qui est la prédiction, une fonction objectif « mean_absolute_error » et l'optimiseur « Adam » pour entraîner le modèle. Afin éviter le surapprentisage on utilise un « EarlyStopping ».

Voici un résumer du réseau de neurone crée avec Keras

```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 507)]             0
_________________________________________________________________
dense (Dense)                (None, 507)               257556
_________________________________________________________________
dense_1 (Dense)              (None, 350)               177800
_________________________________________________________________
dense_2 (Dense)              (None, 250)               87750
_________________________________________________________________
dense_3 (Dense)              (None, 170)               42670
_________________________________________________________________
dense_4 (Dense)              (None, 60)                10260
_________________________________________________________________
dense_5 (Dense)              (None, 20)                1220
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 21
=================================================================
Total params: 577,277
Trainable params: 577,277
Non-trainable params: 0
_________________________________________________________________
```

FIGURE 4.8 – Architecture du modèle 3

Après l'entrainement du modèle sur la collection de t'entraiment, on peut afficher le graphe indiquant la diminution de la valeur de la fonction de perte « *loss function* » durant l'entrainement du modèle sur les données de t'entrainement et de validation.

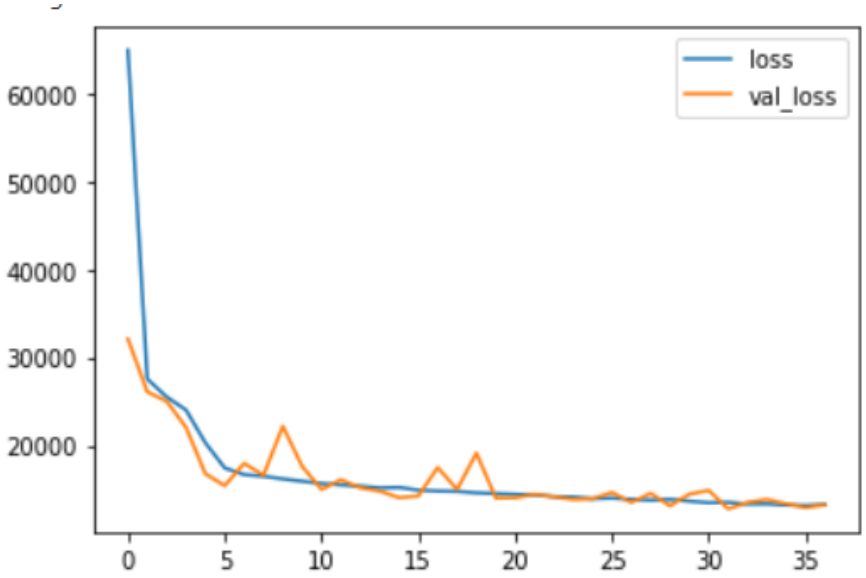


FIGURE 4.9 – Evolution de l'erreur du modèle 3 durant l'entrainement

Après l'entrainement du modèle, Nous pouvons évaluer sa précision sur la collection de test est cela en calculant des mesures de performances. Les résultats obtenus sont décris dans la figure 4.10.

```
RMSE : 32890.52254984386
MAE : 12778.189844455266
EVS : 0.9007731630001794
```

FIGURE 4.10 – calcule des performances du modèle 3

Le graphe suivant « figure 4.11 » indique on rouge les valeurs attendues (prix des propriétés immobilières) et les points bleus sont les prédictions du modèle.

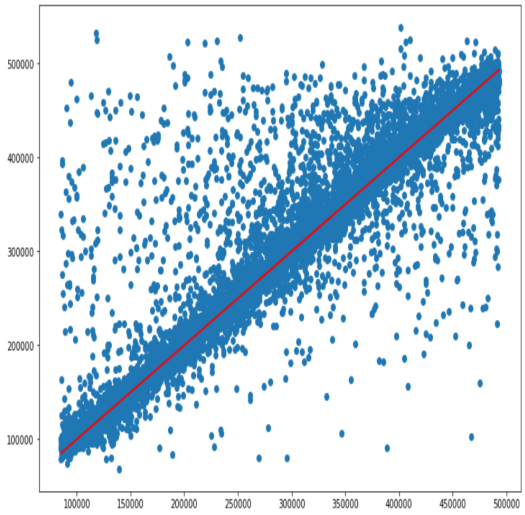


FIGURE 4.11 – valeurs attendues vs prédictions pour le modèle 2

5 Comparaison des résultats

La figure 4.12 montre un tableau récapitulatif des résultats des trois modèles entrainés sur la même collection de données.

Pour commencer, nous observons très nettement que l'on a obtenu de meilleurs résultats avec le modèle 3 à base de réseaux de neurones, en particulier pour la mesure *Mean Absolute Error* de 12778, c'est à dire qu'en moyenne le modèle est à 12778 $ du prix réel de la maison. Concernant les modèles 1 et 2 nous obtenons des résultats assez similaires pour les trois mesures de performances, néanmoins le modèle 2 prend énormément de temps pour son entrainement comparant aux deux autres modèles.

Mesures / *Modele*	Root Mean Square Error	Mean Absolute Error	Explained Variance Score	temps d'entrainement
Modèle 1	32118.6102	14985.1681	0.9036856819671103	890.3071s
Modèle 2	32959.3360	14608.0686	0.898576783484554	9456.5185s
Modèle 3	32890.5225	12778.1898	0.9007731630001794	516.5228s

FIGURE 4.12 – Comparaison des résultats des trois modèles

La figure 4.13 montre le nombres d'échantillons dans notre collection selon le prix des maisons, nous pouvons voir qu'il y a très peu d'échantillons pour les maisons ayant un prix variant entre [450000-500000], et en visualisant les figures 4.4, 4.7, 4.11, nous pouvons remarquer que le modèle 3 à base de réseaux de neurones arrive à avoir des estimations bien plus proches du prix réel comparant aux modèles 2 et 3 qui on des estimations bien plus éloignés du prix réel concernant cette intervalles de prix bien plus chers que la moyenne. Donc, nous pouvons dire que le modèle 3 arrive à avoir des estimations assez proches du prix réel malgré un nombre d'échantillons d'entrainement bien plus bas. Concernant les modèles 1 et 2 nous remarquons donc qu'ils ont du mal à avoir une bonne estimation concernant des maisons ayant un prix plus élevé que la moyenne et cela est du aux faibles nombres d'échantillons d'entrainement.

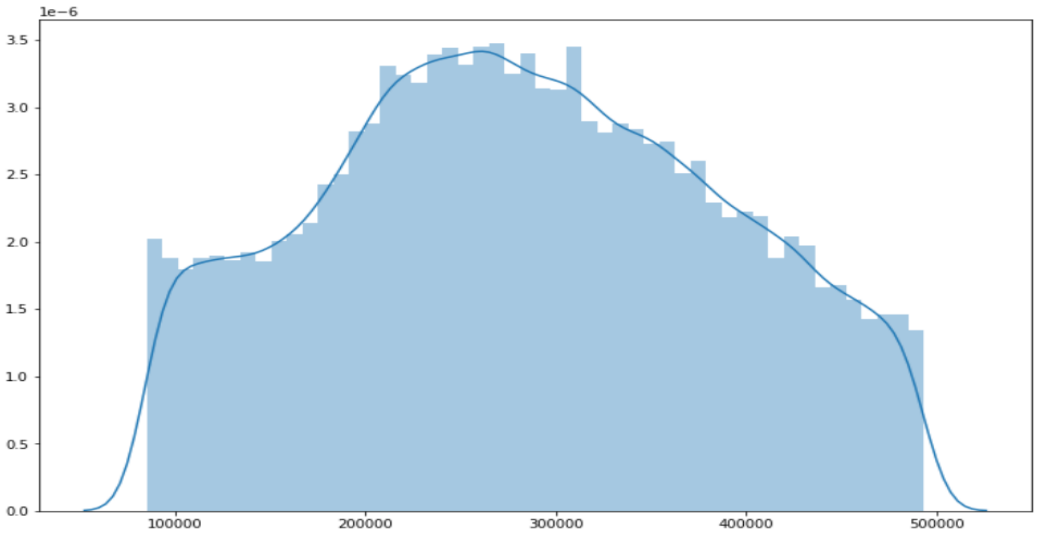


FIGURE 4.13 – Nombres d'échantillons selon les prix des maisons

6 Conclusion

Dans ce dernier chapitre, nous avons présenté dans un premier temps, les différentes bibliothèques python utilisées dans le domaine de l'apprentissage automatique ainsi que l'utilité de chacune d'elles. Par la suite, nous avons montré la manière d'effectuer les différentes étapes de prétraitement sur la collection de données immobilière, suivie par la création des collections d'entrainement et de test. Puis, nous avons créé les trois modèles proposés que nous avons entrainé puis tester. Enfin, nous avons effectué une comparaison des résultats de ces trois modèles.

La comparaison des résultats des trois modèles ont abouti à plusieurs constatations. Dans un premier temps, le modèle 3 à base de réseaux de neurones donne les meilleurs résultats durant son test comparant aux deux autres modèles, il est également plus rapide durant son entrainement. Dans un deuxième temps, en examinant les figures 4.4, 4.7, 4.11, 4.13, on peut remarquer que les modèles proposés ont plus de mal à avoir des prédictions acceptables pour ce qui est des maisons ayant un prix bien plus élevé que la moyenne et cela est dû aux faibles nombres d'échantillons disponibles pour cette catégorie de maison. Ainsi, nous pouvons dire que pour construire un bon modèle d'apprentissage automatique effectuant de l'estimation immobilière, il nous faut des collections de données avec assez d'échantillons pour chaque tranche de prix, afin que le modèle puisse donner des estimations acceptables pour chacune des tranches.

Conclusion générale

Synthèse :

L'apprentissage automatique prend racine et fait ses preuves dans divers domaines. Les travaux présentés dans ce mémoire se penchent sur l'utilisation de cette branche de l'informatique appliquée à l'estimation de la valeur marchande de propriétés immobilière. En effet, Nous avons souligné l'importance de l'industrie immobilière qui est une industrie de base importante pour le développement de l'économie nationale. C'est aussi une industrie pilier qui favorise le développement régulier de l'économie régionale. Néanmoins, les facteurs macro-environnementaux tels que la stabilité politique, la politique fiscale, les contraintes environnementales, etc et la rareté des données ne sont que quelques-unes des raisons pour lesquelles l'évaluation des biens demeure une tâche difficile. Ainsi, il est important de tester différents techniques pour construire des modèles plus performants pouvant obtenir de meilleures estimations afin de faire progresser ce domaine. En effet, ces dans cette optique que se sont articulées nos contributions.

Pour bien cerner notre thème, nous avons choisi de répartir notre travail en quatre chapitres. Dans le premier, nous présentons globalement ce qu'est l'apprentissage automatique. Le second chapitre du travail est consacré à une présentation de l'apprentissage par réseaux de neurones dans le domaine de l'apprentissage profond. Le troisième chapitre est organisé autour de trois points. Nous présentons dans un premier temps un état de l'art de l'estimation immobilière. Nous nous intéressons dans un second temps aux différentes techniques de prétraitement des données utilisées pour préparer les données d'entré des modèles. Ensuite nous décrivons dans le troisième point trois modèles proposés pour l'estimation immobilière. Le quatrième chapitre est consacré à l'implémentation et à l'évaluation des modèles proposés. Nous tentons de mettre en relief les points suivants : la présentation des différentes bibliothèques Python que nous avons utilisé, les prétraitements que nous avons effectués sur la collection de données immobilière, la création des trois modèles proposés que nous entrainons puis testons afin d'obtenir des résultats. Enfin, nous comparons les résultats obtenus pour les trois modèles proposés.

La comparaison des résultats nous amène à dire que le modèle 3 à base de réseaux de neurones donne de meilleurs estimations comparant aux deux autres modèles. Nous remarquons aussi que les trois modèles ont tendances à avoir de mauvaises estimations pour les tranches de maisons plus chers que la moyenne et cela est du aux faibles nombres d'échantillons.

Perspectives :

Nous envisageons certaines perspectives pour le travail présentés tout au long de ce mémoire, nous en décrivant quelques-unes :

- Intégrer le modèle dans une plateforme Web à l'instar du modèle Zestimate du site web Zillow.
- Une deuxième perspective consisterait à construire une collection de données immobilières contenant d'une part un nombre d'échantillons acceptable pour chacune des tranches de prix. D'autre part, la collection pourrait contenir en plus des caractéristiques des maisons, les prix antérieurs de ces biens afin de construire un modèle prenant en compte la temporalité.

Bibliographie

Bibliographie

[1] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow : Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2019.

[2] I. Vasilev, D. Slater, G. Spacagna, P. Roelants, and V. Zocca, *Python Deep Learning : Exploring deep learning techniques and neural network architectures with Pytorch, Keras, and TensorFlow*. Packt Publishing Ltd, 2019.

[3] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*. Springer, 2013, vol. 112.

[4] G. Shepherd, "Koch. c," *The synaptic organization of the brain. Oxford University Press, New York*, 1990.

[5] C. C. Aggarwal *et al.*, *Neural networks and deep learning*. Springer, 2018.

[6] J. Moolayil, J. Moolayil, and S. John, *Learn Keras for Deep Neural Networks*. Springer, 2019.

[7] F. Chollet, *Deep Learning mit Python und Keras : Das Praxis-Handbuch vom Entwickler der Keras-Bibliothek*. MITP-Verlags GmbH & Co. KG, 2018.

[8] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1.

[9] F. Chollet, *Deep Learning with Python*. Manning, Nov. 2017.

[10] B. J. Kim, "Improved deep learning algorithm," *JOURNAL OF ADVANCED INFORMATION TECHNOLOGY AND CONVERGENCE*, vol. 8, no. 2, pp. 119–127, 2018.

[11] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of research and development*, vol. 3, no. 3, pp. 210–229, 1959.

[12] T. M. Mitchell, "Machine learning, volume 1 of 1," 1997.

[13] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[14] L.-P. Chen, "Mehryar mohri, afshin rostamizadeh, and ameet talwalkar : Foundations of machine learning," 2019.

69

[15] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*. Springer series in statistics New York, 2001, vol. 1, no. 10.

[16] S. Raschka and V. Mirjalili, *Python machine learning*. Packt Publishing Ltd, 2017.

[17] B. T. Polyak, "Some methods of speeding up the convergence of iteration methods," *USSR Computational Mathematics and Mathematical Physics*, vol. 4, no. 5, pp. 1–17, 1964.

[18] Y. Nesterov, "A method for unconstrained convex minimization problem with the rate of convergence o (1/kˆ 2)," in *Doklady an ussr*, vol. 269, 1983, pp. 543–547.

[19] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of machine learning research*, vol. 12, no. 7, 2011.

[20] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop : Divide the gradient by a running average of its recent magnitude," *COURSERA : Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.

[21] D. P. Kingma and J. Ba, "Adam : A method for stochastic optimization," *arXiv preprint arXiv :1412.6980*, 2014.

[22] G. Favara and Z. Song, "House price dynamics with dispersed information," *Journal of Economic Theory*, vol. 149, pp. 350–382, 2014.

[23] K. J. Borowiecki *et al.*, "The determinants of house prices and construction : an empirical investigation of the swiss housing economy," *International Real Estate Review*, vol. 12, no. 3, pp. 193–220, 2009.

[24] S. Das, R. Gupta, and A. Kabundi, "Could we have predicted the recent downturn in the south african housing market ?" *Journal of Housing Economics*, vol. 18, no. 4, pp. 325–335, 2009.

[25] O. Špačková and D. Straub, "Dynamic bayesian network for probabilistic modeling of tunnel excavation processes," *Computer-Aided Civil and Infrastructure Engineering*, vol. 28, no. 1, pp. 1–21, 2013.

[26] Y. Huang, J. L. Beck, S. Wu, and H. Li, "Robust bayesian compressive sensing for signals in structural health monitoring," *Computer-Aided Civil and Infrastructure Engineering*, vol. 29, no. 3, pp. 160–179, 2014.

[27] B. Égert and D. Mihaljek, "Determinants of house prices in central and eastern europe," *Comparative economic studies*, vol. 49, no. 3, pp. 367–388, 2007.

[28] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[29] H. Crosby, P. Davis, T. Damoulas, and S. A. Jarvis, "A spatio-temporal, gaussian process regression, real-estate price predictor," in *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2016, pp. 1–4.

[30] J. Li, J. Wang, and S. Zhou, “A study on non-traditional factors affecting price differences of real estate in different regions-based on 35 chinese cities panel data,” in *Proceedings of the 4th International Conference on Industrial and Business Engineering*, 2018, pp. 1–8.

[31] J. Niu and P. Niu, “An intelligent automatic valuation system for real estate based on machine learning,” in *Proceedings of the International Conference on Artificial Intelligence, Information Processing and Cloud Computing*, 2019, pp. 1–6.

[32] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[33] H. Dai, G. Xue, and W. Wang, “An adaptive wavelet frame neural network method for efficient reliability analysis,” *Computer-Aided Civil and Infrastructure Engineering*, vol. 29, no. 10, pp. 801–814, 2014.

[34] B. A. Story and G. T. Fry, “A structural impairment detection system using competitive arrays of artificial neural networks,” *Computer-Aided Civil and Infrastructure Engineering*, vol. 29, no. 3, pp. 180–190, 2014.

[35] A. Khalafallah, “Neural network based model for predicting housing market performance,” *Tsinghua Science and Technology*, vol. 13, no. S1, pp. 325–328, 2008.

[36] H. Selim, “Determinants of house prices in turkey : Hedonic regression versus artificial neural network,” *Expert systems with Applications*, vol. 36, no. 2, pp. 2843–2852, 2009.