Rapport final - Algorithmes sans biais pour le recrutement

Projet 40 :

Bennis Jihane, Guibert Rachel, Than Van Con Marc

March 2019

Table des matières

1 Remerciements

Nous tenons à remercier tout particulièrement Céline Hudelot pour nous avoir accompagnés pendant ce projet ainsi que l'ensemble des membres du MICS qui nous ont fourni les moyens de mener à bien nos travaux. Nous remercions aussi l'entreprise Randstad pour les données fournies sans lesquelles ces travaux auraient été impossibles.

2 Introduction

Depuis quelques années, les algorithmes de machine learning sont de plus en plus utilisés dans des usages de la vie quotidienne et on peut présumer sans trop se tromper que cette tendance n'ira pas en diminuant. Non content de toucher de plus en plus de monde, ces algorithmes concernent de plus en plus des domaines critiques de l'activité humaine. De ce fait, ils sont amenés à prendre des décisions avec de forts enjeux éthiques et un impact de plus en plus grand sur les personnes, conduisant à la création de nouvelles réflexions dans le domaine, rassemblées sous le nom de "Fairness (justice) in Machine Learning". Un des sujets importants de cette nouvelle discipline est de questionner la reproduction (voir l'amplification) des biais humain par les algorithmes de Machine Learning. En effet, pour entraîner de nouveaux algorithmes, il faut s'appuyer sur des données déjà existantes et généralement biaisées, qui conduisent ensuite les algorithmes à produire eux mêmes des décisions biaisées par la suite. Il a en effet été montré que les algorithmes reproduisaient des biais implicites et inhérents aux données, notamment en ce qui concerne les corpus de texte. Ces biais peuvent ensuite avoir des conséquences très grave :on sait maintenant par exemple qu'un des logiciels utilisés par la justice américaine pour juger les demande de sursis donne un taux de récidive 75% plus important aux personnes noires qu'aux personnes blanches à crime égal, ce qui questionne bien entendu sur les décisions qui peuvent être ensuite prise.

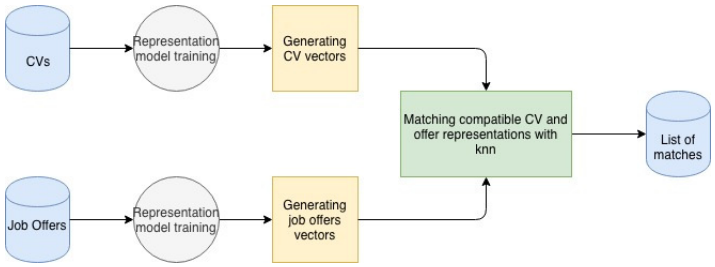
Le recrutement est l'un des domaines où le machine learning peut avoir des applications très intéressantes : en effet, pour un certains nombre d'entreprises, il y a un réel enjeu économique à mettre en correspondance efficacement (c'est à dire rapidement et avec un haut taux de réussite) des candidats et des offres, ce que permettent certains algorithmes bien entrainés. La question de la réduction du biais est alors de la plus grande importance comme on a pu le voir récemment, lorsque Amazon s'est trouvé obligé de retirer leur algorithme de recrutement, qui, reproduisant une situation existante dans l'entreprise, favorisait les hommes pour des positions d'ingénieurs. Dans le cadre de ce projet nous allons donc nous pencher sur la réduction du biais dans le domaine du recrutement. Nous allons uniquement nous intéresser aux données de type CV et aux biais de genre dans le cadre de ce projet, mais celui ci s'inscrit plus globalement dans un projet de création d'un algorithme de recrutement non biaisé, où on pourra prendre en compte différents types de biais (origine,...) et différents types de données (les entretiens par exemple).

3 Chaîne de traitement du biais dans les CVs

3.1 Schéma d'ensemble

Après avoir effectué notre recherche bibliographique nous avons découvert qu'il existait plusieurs techniques de mise en correspondance de candidatures et d'offres d'emplois. Celle que nous avons décidé de traiter consiste à extraire, avec des réseaux de neurones profonds, des représentations vectorielles des compétences et expérience présentes dans les CVs ; puis faire de même pour les exigences présentes dans les offres d'emplois. La technique des plus proches voisins est ensuite utilisée pour mettre en correspondance les deux représentations vectorielles obtenues.

FIGURE 1 – Schéma du processus de mise en correspondance étudié



3.2 Description des spécifications fonctionnelles

Dans cette approche de mise en correspondance, nous avons décidé d'intervenir sur l'extraction du contenu des CVs. Les réseaux de neurone nous permettent d'extraire de l'information, à savoir les compétences et les expériences des candidats, sous forme d'une représentation vectorielle. Nous voulons mettre en place un système de génération de représentations vectorielles de CV non biaisées au niveau de genre.

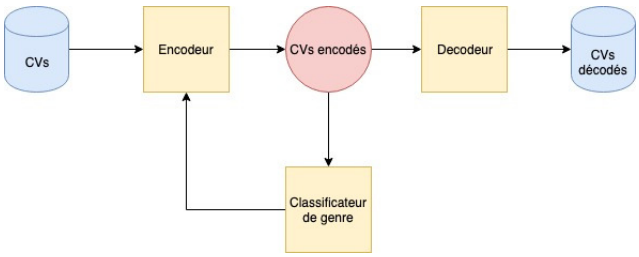
3.3 Méthode de traitements

Pour cela, nous avons décidé d'adopter une approche qui s'inspire des Generative Adversarial Networks (GANs) et les autoencoders. Les GANs sont des réseaux neuronaux utilisés notamment pour générer à partir d'entrées quelconques des images réalistes. Une des applications les plus connues de cette technique est le site "This person does not exist". Sur ce site sont exposés des photos de personnes qui n'existent pas généré à l'aide de GANs. Dans un GAN, deux réseaux de neurones sont entraînés simultanément. Le premier réseau est ce qu'on appelle un générateur et est entrainé à générer des images à partir d'images d'entrainement. Le deuxième réseau est ce qu'on appelle un discriminateur. Le discriminateur est un classificateur dont le but est de dire si l'image produite par le générateur est une image fausse ou réelle. Cette classification est alors prise en compte dans les phases d'entrainement suivantes du générateur. Grâce à cet entrainement simultané du générateur et du discriminateur, le générateur parvient à la fin à réaliser des images "fausses" qui sont labellisées comme vraies par le discriminateur.

Les autoencoders sont eux aussi des réseaux neuronaux reposant sur deux sous-réseaux eux-même neuronaux : un encodeur et un décodeur. Le but de ces réseaux est dans un premier temps de réduire la dimensionnalité des entrées initiales et de les encoder dans des vecteurs de plus petites taille et dans un deuxième temps retrouver l'entrée initiale à partir de ce nouveau vecteur,

de le décoder.

FIGURE 2 – Schema du processus d'élimination du biais



Notre but est de faire une synthèse de ces deux approches. Nous devons en effet répondre à une double exigence. La première énoncée précédemment est celle d'apprendre une représentation non biaisée des CVs, la deuxième est de faire en sorte que l'on apprenne également les compétences présentes dans les CVs et qu'elles ne soient pas perdues dans le processus d'élimination du biais de genre. Afin de répondre à cette double exigence nous optons donc pour l'entrainement d'un autoencoder dont le but est d'encoder puis de restituer le CV donné et d'entrainer en parallèle un classificateur de genre sur la sortie de la partie encodeur de la chaine. En fin d'entrainement nous espérons donc obtenir un encoder qui n'apprend pas le genre mais qui conserve les informations présentes dans le CV.

4 Implémentation de la chaîne de traitement

Toute la chaîne de traitement a été implémentée en Python. Ce langage est en effet en croissance dans le monde et détient une communauté très active et une documentation complète. Il est de plus très utilisé dans le domaine du Machine Learning et tout particulièrement dans le Deep Learning. Dans un souci de postérité, nous avons naturellement opté pour ce langage.

4.1 Pré-traitement des CVs

4.1.1 Labellisation des CVs

Après avoir récupéré les CVs nous nous sommes aperçus que contrairement à ce que nous pensions, les CVs n'étaient pas labellisés pour le genre. Vu le nombre de données (environ 20 000 CVs) nous nous sommes rapidement rendus compte qu'il allait être impossible de réaliser une labellisation manuelle de nos données. Nous avons donc dû créer une chaine de traitement pour assurer une labellisation correcte d'un grand nombre de ces données. Pour celà, nous avons décidé d'utiliser le fait que le prénom des personnes apparaît généralement dans les premiers mots d'un CV, et que la plupart des prénoms en français étaient "genrés" i.e. associés à plus de 75% à un genre. Nous avons utilisé une base de prénoms de l'INSEE, qui pour chaque prénom et chaque année entre 1900 et 2017, associe le nombre d'occurence du prénom pour chaque genre. A partir de cette base de données, nous avons pu calculer avec quelles proportions les prénoms avaient été données à des hommes ou des femmes entre les années 1950 et 2000 (population active actuelle), et en associer un certain nombre au genre féminin (resp. masculin), s'ils étaient données à plus de 75% à des femmes (resp. à des hommes).
Nous avons ensuite comparés les 20 premiers mots de tous les CVs, excluant les noms de rue (le prénom se trouvant généralement avant) à notre bas de données de noms genrés. Quand la réponse était non équivoque (c'est à dire qu'il n'y avait pas deux noms ou plus de genres différents dans ces 20 mots), le CV était associé à cette réponse.

Cette labellisation n'est bien entendu pas parfaite : nous avons volontairement accepté une réduction de notre dataset à 63% de notre dataset initial pour éviter les cas contentieux. Ainsi, l'ensemble des Dominique (nom non genré) ou des Marie Benoit (deux noms de genre différents) a été exclue d'office de notre dataset. Néanmoins, nous sommes plutôt satisfait de cette solution, qui nous permet de garder près de 12000 CVs labellisés pour la variable souhaitée.

4.1.2 Vectorisation des CVs

Pour pouvoir établir une chaîne d'apprentissage sur nos données de CVs, il a fallu trouver une base commune pour la représentation des CVs.

Pour ce projet, nous avons décidé de nous focaliser principalement sur les données textuelles d'un CV. La représentation du contenu textuel dans la littérature est ainsi traitée de différentes façons.
D'abord, nous avons pré-traité les données en supprimant la ponctuation et les *stop words*, c'est-à-dire les mots qui n'apportent théoriquement pas de valeur supplémentaire, comme les conjonctions de coordination, les articles *et, à, le, la etc..* en utilisant la bibliothèque *nltk*.
Puis, nous avons décidé d'explorer deux manières de représentation parmi les plus connues :

1. **Sac de mots ou *bag of words***

— **Principe :**
Cette technique est très utilisée en recherche d'information. Le principe est simple : on considère qu'un document peut être représenté selon un *dictionnaire* de mots. On appelle ce dictionnaire un vocabulaire. Chaque document, ici CV, est représenté comme vecteur de la taille du vocabulaire et sa composante i indique le nombre d'occurrences du *i-ème* mot du dictionnaire au sein du document. Par exemple, considérons :
vocabulary = *['Modélisation', 'Data', 'Mathématiques', 'électron']*
document=['Modélisation de la dynamique des fluides. Modélisation du mouvement des électrons.']
Alors on ce document sera représenté par le vecteur : *representation=[2, 0, 0, 1]*
— **Implémentation :** On a utilisé la librairie de *Sickit-learn* pour la vectorisation *bag of words*. Cette librairie met à disposition un *CountVectorizer* permettant de compter le nombre de mots uniques
— **Limitations :**
Cette approche a deux limitations majeures. En effet, le Sac de Mots ne prend pas en compte la signification d'un mot. Le contexte dans lequel un mot est utilisé n'a pas d'importance, ce qui peut être problématique étant donné qu'un CV est une suite de phrases. De plus, cette méthode génère des vecteurs de la taille du vocabulaire considéré. Au vue de la quantité de données, cela devient vite problématique en terme de besoin en puissance de calcul.

2. **Tokénisation en mots**

— **Principe :** La tokénisation d'un texte consiste à le découper en mots. De manière similaire à un *bag of words*, cette technique repose sur la représentation d'un texte selon les mots qui le constituent. Mais cette méthode permet de fixer deux paramètres importants : la taille du vocabulaire *num-words* et la taille du vecteur représentant le document *len-max-sequence*. Nous représentons ainsi le corpus par des vecteurs de taille *len-max-sequence* composés des indexes de mots générés à partir du vocabulaire.
— **Implémentation :** Nous avons implémenté cette méthode en utilisant la bibliothèque *keras*. Les paramètres *num-words* et *max-len-sequence* ont été déduit d'une étude de l'état de l'art et pourront faire l'objet d'une étude plus spécifique pour améliorer le modèle. La librairie *keras*, et plus spécifiquement la sous-librairie *preprocessing* permet ainsi de générer un modèle de tokénisation. Nous avons en effet appliqué un Tokenizer s'entraînant sur une partie de nos données afin qu'il apprenne le vocabulaire. On peut ainsi appliquer ce modèle à un nouveau CV, et le représenter selon le vocabulaire généré sous forme d'un vecteur de taille fixe *max-len-sequence*.
— **Limitations :** De même que *bag of words*, la tokénisation perd le contexte d'un mot au sein de sa phrase. De plus, les paramètres *num-words* et *max-len-sequence* peuvent compromettre l'efficacité de la représentation.

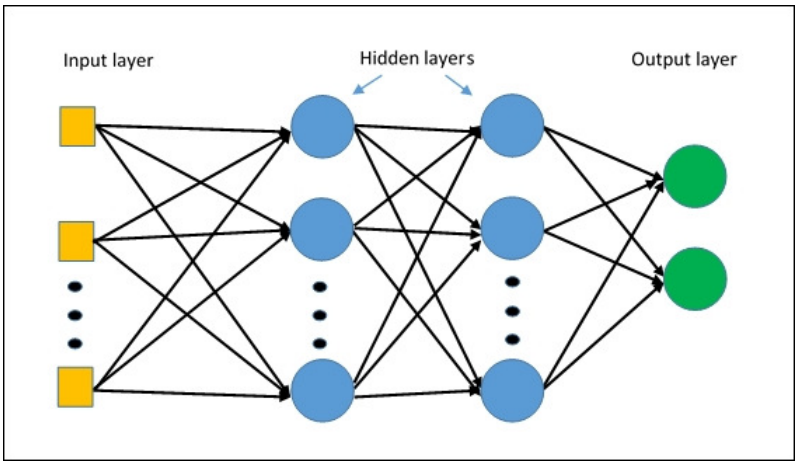
Pour la suite du projet, nous avons opté pour la méthode *Tokénisation en mots* qui promettait une meilleure généralisation et de meilleures performances par rapport à *bag of words*.

4.2 Classificateur de genre

Une fois le pré-traitement des CVs effectué, nous avons implémenté les différentes parties du code nécessaires pour le "débiaisement" des données - principalement en *Keras*, framework de *TensorFlow*.
Pour la classification du genre, nous avons décidé d'implémenter un **perceptron multicouches**

(MLP).
Il s'agit d'un type de réseau neuronal formel organisé en plusieurs couches au sein desquelles une information circule de la couche d'entrée vers la couche de sortie uniquement. L'intérêt d'un MLP est que ses couches sont totalement connectées, ce qui permet de préserver toute l'information mais peut demander des ressources en calcul plus grandes. Étant donné que nous ne considérons que l'aspect textuel des CVs et que nous n'avons que 12000 CVs, la capacité de calcul ne nous a pas posé problème. La structure générale d'un MLP s'illustre de la sorte :

FIGURE 3 – Schéma général d'un MLP



Pour le classificateur de genre implémenté, les différentes composantes du MLP sont les suivantes :

- *Entrée :* La représentation tokénisée des CVs. D'un point de vue plus pratique, l'entrée est composée d'un vecteur de taille 1024, qui correspond à la longueur que nous avons fixée pour la représentation de chaque CV.
- *Couche Cachée :* Nous avons opté pour une couche d'activation ReLu, largement utilisée dans l'état de l'art.
- *Couche de sortie :* Nous avons décidé d'appliquer une normalisation *softmax* et avons fixé la forme de sortie à un vecteur de deux dimensions. La sortie correspond ainsi à un vecteur *(male, female)* contenant la prédiction du réseau neuronal sous la forme d'une probabilité pour chacune des deux classes.
- *La fonction de coût :* Nous avons également choisi une fonction de coût largement utilisée dans le domaine de la classification : la *cross entropy*

Afin de mesurer l'efficacité de notre classificateur, nous nous sommes intéressés à la métrique *accuracy*. Nous discuterons les résultats en section *5*.

4.3 Auto-encoder

L'implémentation d'un auto-encodeur consiste en l'implémentation de deux réseaux neuronaux. Nous avons décidé d'implémenter deux MLP à une seule couche cachée, chacun avec des entrées/sorties correspondant à sa fonctionnalité. Ainsi, nous avons :

- *Réseau neuronal de l'encodeur :* ce MLP a pour entrée le CV tokénisé, soit un vecteur de dimension 1810 (*taille fixée lors du preprocessing*) et une sortie correspondant à la

représentation matricielle des CVs, de taille 1024.

— *Réseau neuronal du décodeur :* ce MLP prend en entrée la représentation matricielle générée par l'encodeur, de taille 1024. Sa sortie a les dimension du CV tokénisé, soit une dimension de 1810.

Chacun de ces réseaux neuronaux a été implémenté avec une fonction d'activation ReLu, et une fonction de coût sous forme d' **erreur quadratique moyenne** ; un choix basé sur l'état de l'art des pratiques en apprentissage profond.

4.4 Boucle adversariale

Enfin, la dernière partie de l'implémentation est la boucle adversariale.
Afin de la réaliser, nous avons procédé de la sorte :

1. Tout d'abord, nous avons ajouté **une fonction de coût caractéristique de la boucle adversariale**. Au sein de cette boucle, nous prenons en compte les fonctions de coût de l'autoencodeur et celle du classificateur. Le but est de minimiser le coût de l'autoencodeur, pour s'assurer qu'il restitue le plus d'information possible, et en parallèle forcer la fonction de coût du classificateur à augmenter, ce qui nous permet de diminuer les performances du classificateur. En effet, rappelons-le, le but de la boucle adversariale est d'obtenir un autoencodeur le plus efficace possible, générant une représentation (*sortie de l'encodeur*) dont le genre est indiscernable - ce qui se traduit par une médiocre performance du classificateur de genre sur la représentation en cours de débiaisement. La fonction t adversariale adoptée est donc de la forme :

$$Loss_{adversarial} = Loss_{autoencoder} - \beta * Loss_{classifier}$$

où β est un paramètre, $Loss_{autoencoder}$ et $Loss_{classifier}$ les fonctions de coût de l'autoencodeur et du classificateur.

2. Ensuite, lors de l'entraînement du modèle, nous nous sommes assurés d'**alterner l'apprentissage** de l'autoencodeur et de celui du classificateur. Par exemple, on entraîne l'autoencodeur une fois sur deux, on en profite pour mettre à jour la fonction de coût adversariale, puis on entraîne le classificateur sur les nouvelles représentations générées par l'encodeur. *La fréquence d'alternance est un paramètre que l'on étudiera plus tard.*

4.5 Entraînement des modèles

L'entraînement de nos modèles a été implémenté avec les *Session* de *TensorFlow*. Afin de mener nos entraînements, nous avons séparé nos données labelisées en trois ensembles :

— ***Ensemble d'entraînement*** : Les données sur lesquelles les modèles s'entraînent pour mettre à jour leurs poids.

— ***Ensemble de Validation*** : Des données sur lesquelles nous testons les modèles durant l'entraînement. Cela nous permet de vérifier la généralisation du modèle ainsi que sa performance durant l'apprentissage.

— ***Ensemble de Test*** : Des données sur lesquelles nous appliquons le modèle à la fin de l'entraînement. Cela nous permet d'avoir une idée globale de la généralisation et des performances des modèles sur des données qu'il n'a jamais encore rencontrées.

L'entraînement sur les données se fait par *batch* afin d'éviter un sur-apprentissage de l'ensemble d'apprentissage.

Comme expliqué précédemment, l'entraînement des modèles se fait par alternance : l'autoencodeur puis le classificateur. Au fil de l'entraînement, nous affichons ainsi les *Loss* de l'autoencodeur et de la boucle adversariale, et l'Accuracy du classificateur pour bien suivre les performances de chaque architecture choisies. Nous avons en effet construit différents modèles en jouant sur différents hyperparamètres : β, *taille du batch, Pas d'apprentissage des modèles, Fréquence d'alternance de l'apprentissage des modèles* .
Nous allons observer les conséquences de chacun de ces hyperparamètres en section *5* de ce rapport.

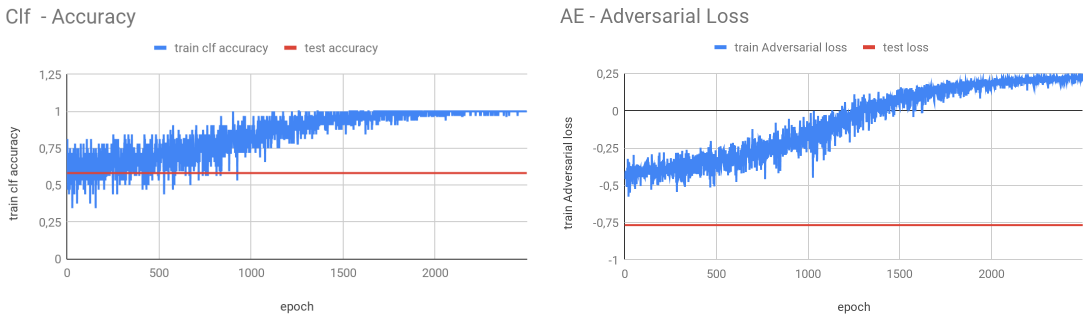
5 Analyse des résultats et limites de nos travaux

Lors de l'entraînement et du test de nos modèles, nous avons été rapidement été confrontés à deux problématiques :

— Nous avions en général de gros problèmes d'**overfitting** sur l'entraînement de nos modèles, comme le montre la figure *4.a*. En effet, on s'aperçoit d'abord l'accuracy du classificateur ne cesse d'augmenter pour atteindre 1 pendant l'entraînement, ce qui est contraire à notre besoin, mais aussi que cette valeur ne correspond pas du tout à la valeur de test, quasiment situé à l'aléatoire, ce qui est un bon signe d'overfitting.

— Dans la figure *4.b*, on voit un cas où le "débiaisement" n'est pas efficace. En effet, la loss de notre modèle est croissante, alors qu'on la souhaiterait décroissante et si possible négative.

Pour résoudre ces problématiques, nous avons cherché comment les différents hyperparamètres influençaient ces phénomènes, et quels seraient les meilleurs à adopter.



(a) Exemple d'overfitting

(b) Débiaisement non efficace

FIGURE 4 – Les problématiques du modèle

5.1 Influence de la taille du batch

Les figures *5* montrent bien que la taille du batch a une influence majeure sur l'overfitting : plus le batch est grand, plus le risque d'overfitting est important. Lorsque le batch est égal à 10, on a par contre une situation assez intéressante : la loss de notre modèle adversarial est quasi constante et proche de la valeur de test. On voit aussi dans ce cas que cette stabilité n'implique pas nécessairement une stabilité de l'accuracy : on est dans un système assez fortement oscillant, mais qui est centré sur la valeur de test, ce qui est assez intéressant pour nous. Le système a alors une certaine **stabilité**.

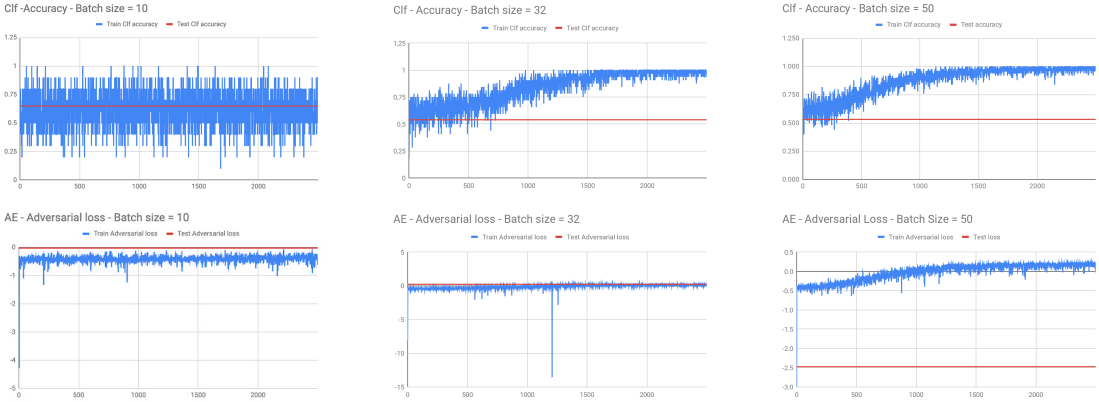


FIGURE 5 – Influence de la taille du batch

5.2 Influence du pas d'apprentissage

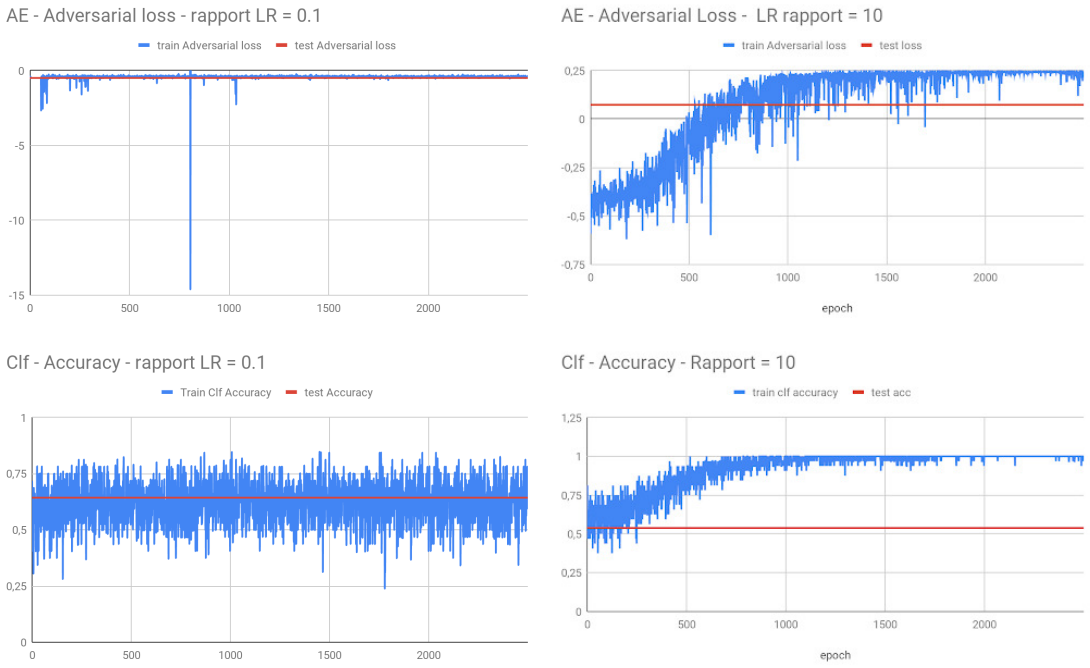


FIGURE 6 – Influence du ratio des pas d'apprentissage

Le deuxième paramètre dont nous avons étudié l'influence est le rapport des pas d'apprentissage du classifieur et de l'autoencodeur. $LR = \frac{Pas\ d'apprentissage\ de\ l'autoencodeur}{Pas\ d'apprentissage\ du\ classifieur}$. On remarque que lorsque ce rapport est grand, on a de l'overfitting, ce qui s'explique assez aisément par le fait que le classifieur avec un pas plus petit est beaucoup plus sensible et "apprend mieux". On choisit donc les pas d'apprentissage de telles manières à avoir $Pas\ d'apprentissage\ de\ l'autoencodeur \leq$

Pas d'apprentissage du classifieur.

5.3 Influence du paramètre β

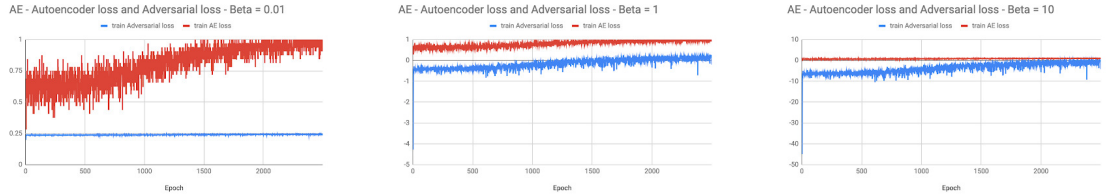


FIGURE 7 – Influence du paramètre β

Nous avons aussi étudié l'importance du paramètre β sur le comportement de notre modèle. POur celà, nous avons étudié l'évolution de la loss de la boucle adversariale d'une part, et de l'autoencodeur d'autres part. En effet, l'objectif est à la fois de diminuer la loss de la boucle adversariale, sans amener une trop grosse perte d'information, c'est à dire une loss de l'autoencodeur trop importante. Cet objectif nous permet rapidement d'éliminer le cas de la figure *7.c* où on voit que la loss de la boucle adversariale ne fait qu'augmenter ce qui montre que l'accuracy de notre classifieur ne fait qu'augmenter elle aussi. La figure *7.a* n'est pas non plus un cas idéal, car la loss de l'autoencodeur augmente beaucoup et n'atteint pas de stabilité. Le cas le plus idéal est celui où $\beta = 1$, car il nous montre que la loss du classifieur est constante et positive, ce qui donne une certaine stabilité au modèle et nous permet deprévoir ces performances.

5.4 Influence de l'alternance d'apprentissage

A partir des cas les plus idéaux que nous avions trouvé, c'est à dire de telle manière que le classifieur soit oscillant autour d'une manière stable et une loss de la boucle adversariale constante et proche de 0, nous avons décidé de jouer sur le nombre d'entrainement du classifieur et de l'autoencodeur, espérant diminuer notre loss et l'oscillation du classifieur. En diminuant le nombre d'entraînement du classifieur, nous avons de manière assez surprenante, une augmentation de la loss de la boucle adversariale et peu d'influence sur les oscillations. Pour le moment, cette piste n'est pas concluante et mériterait plus d'exploration.

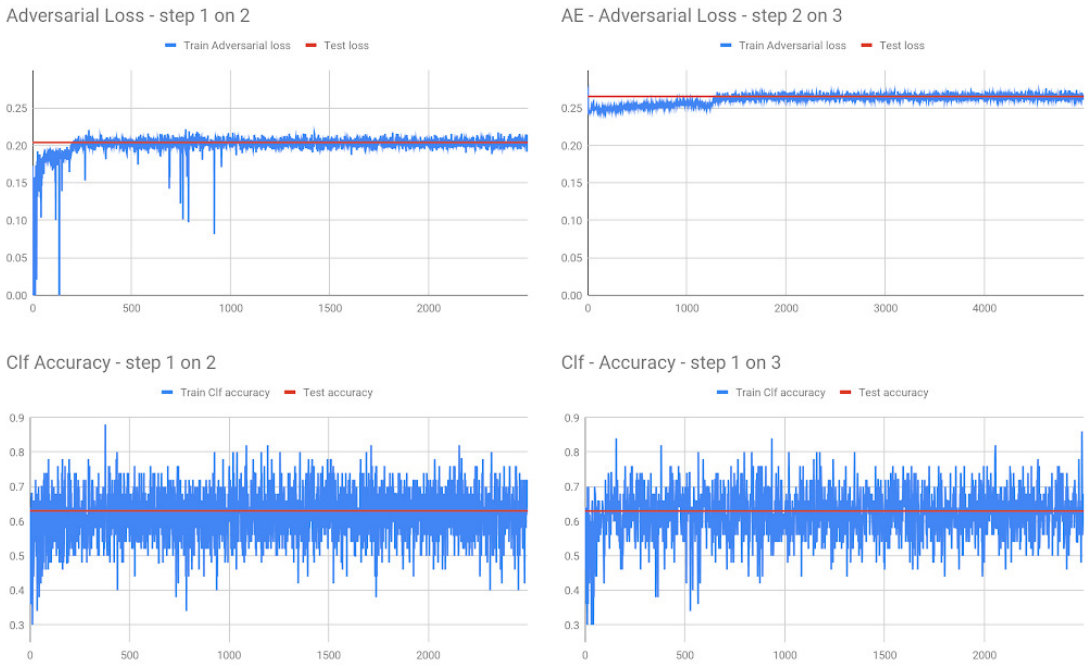


FIGURE 8 – Influence des alternances d'apprentissage

6 Pistes d'amélioration

Au cours de nos travaux nous nous avons essayé différentes combinaisons d'hyperparamètres afin de mieux aboutir à l'élimination du genre des CVs. Dans cette recherche, nous n'avons malheureusement pas pu tester toutes les combinaisons que nous aurions souhaitées. Cette partie présente quelques pistes par lesquelles nos résultats pourraient être améliorés.

6.1 Différentes entrées

Nous avons utilisé durant nos travaux des CVs qui avaient été préalablement parsés de format pdf au format texte. Sur ces documents textuels nous avons effectué une tokenisation et associé à chaque mot un nombre. Chaque mot est alors indexé de 1 à au plus 10000 (taille du vocabulaire que nous avions fixée). Cette tokenisation est simple et peu coûteuse en terme de calcul à réaliser. Cependant elle présente aussi certains désavantages notamment celui de perdre le contexte de chaque mot. D'autres méthodes de pre-encoding peuvent être ici employées.
Une méthode classique utilisée dans le NLP est le *Bag of Words*. Cette méthode consiste à représenter l'ensemble de notre corpus de CVs sous la forme d'une matrice où chaque colonne représente un mot et chaque ligne représente le nombre d'occurrences dans le CV. Nommons cette matrice $A = (a_{i,j})$, chaque coefficient $a_{i,j}$ représente le nombre d'occurrences du mot j dans le CV numéro i. Cependant le Bag of Words présente *deux inconvénients majeurs*. Le premier est la taille des matrices. Ces matrices ont souvent une taille importante et nécessitent donc d'importantes capacités de calcul afin de les utiliser en tant que matériel d'apprentissage. Le deuxième inconvénient est le fait que chaque mot est considéré de manière indépendante des autres. On perd ici l'aspect séquentiel du texte.

La méthode de pre-encoding qui reçoit le plus d'éloge dans les problèmes de NLP est la technique *Word2Vec*. Il s'agit d'une technique de vectorisation à savoir le passage d'un texte à un vecteur comportant des composantes réelles. Word2Vec se base sur un réseau de neurones pré-entraîne qui est ensuite adapté aux corpus de texte que l'on lui soumet. Un des grands avantages de Word2Vec par rapport aux deux méthodes précédemment énoncées est qu'ici le contexte des mots est pris en compte, ce qui est primordial lors de traitement du texte. Ainsi nous recommandons dans le cadre de la poursuite de nos travaux de privilégier la méthode Word2Vec afin d'au mieux vectoriser les CVs. A noter également qu'il existe des réseaux neuronaux pré-entraînés pour le word2vec en français, et les exploiter en faisant du fine-tuning serait une piste très prometteuse.

6.2 Utilisation de réseaux de neurones différents

Qu'il s'agisse de l'autoencodeur ou du classificateur de genre, nous avons utilisé des *Multi Layer Perceptron (MLP)*. Les MLP se distinguent par leur caractère polyvalent et facilement implémentable. Cependant malgré leur polyvalence, les MLP ne sont pas nécessairement les réseaux de neurones offrant les meilleures performances en matière de NLP.

Un type de réseau est particulièrement adapté aux problèmes textuels tels que celui que nous tentons de traiter. Il s'agit des réseaux *LTSM*. Les LSTM sont des réseaux appartenant à la catégorie des Recurrent Neuronal Networks (RNN). Ces derniers sont très adaptés lorsqu'il s'agit de traiter des données séquentielles comme un texte par exemple.

Nous pourrions donc imaginer améliorer les performances de reconstruction et d'oubli du biais de genre en remplaçant tout ou une partie des MLP utilisés dans nos travaux.

6.3 Une fonction de coût différente

Rappelons ici la fonction de coût de notre boucle adversariale :

$$Loss_{adversarial} = Loss_{autoencoder} - \beta * Loss_{classifier}$$

Cette fonction de coût présente l'inconvénient qu'il est difficile d'estimer une valeur idéale pour le coût issu du classificateur. En effet on souhaite que le coût de l'autoencodeur soit proche de 0 et que le coût du classificateur soit élevé, cependant on ne sait pas quelle valeur serait assez élevée pour considérer le classificateur incapable de déceler le genre dans les CVs encodés.

Ce faisant, il serait intéressant d'essayer d'implémenter une boucle reposant sur la fonction de coût suivante :

$$Loss_{adversarial} = Loss_{autoencoder} + \alpha * Accuracy_{classifier}$$

Ici nous avons en tête une valeur idéale pour les deux composantes de notre fonction de coût. Idéalement la fonction de coût de l'autoencoder aurait une valeur proche de 0. Nous voudrions aussi que le classificateur ne puisse pas plus déceler le genre que le hasard, c'est-à-dire que l'Accuracy du classificateur soit proche de 50%. La fonction de coût adversariale tendrait donc idéalement vers $0.5 * \alpha$.

6.4 Hyperparamètres

Nous avons énoncé précédemment une série d'hyperparamètres sur lesquels nous avons expérimenté différentes combinaisons. Nous jugeons nécessaire de mener une recherche approfondie particulièrement sur ces paramètres :

— **Beta :** Il s'agit de l'importance que prend le coût de notre classificateur dans notre fonction de coût. Plus le coefficient β est élevé plus les performances du classificateur ont de l'importance dans notre boucle adversariale . Nous avons essayé différentes valeurs de β qu'elle soient égales, supérieures ou inférieures à 1. Aucune de ces valeurs ne nous a donné satisfaction mais avaient pour conséquence des comportements d'apprentissage différents. Ainsi il serait fortement recommandé d'approfondir cette recherche de la valeur de β idéale.
— **La profondeur du réseau :** Dans notre réseau actuel, l'encodeur, le décodeur et le classificateur ne comportent chacun qu'une seule couche d'activation. Il serait intéressant d'observer les résultats que pourrait avoir l'ajout de couches de neurones dans l'un des trois réseaux. L'ajout de couche est réputé pour améliorer les performances d'apprentissage mais aussi d'augmenter le risque de sur-apprentissage.

6.5 Régularisation

Enfin nous avons vu que nos réseaux étaient très sujets à l'overfitting, ce phénomène qui consiste en de très bonnes performances d'apprentissage mais une très mauvaise capacité de généralisation (résultats de test bien en deçà de ceux observés pendant d'entraînement). L'overfitting est un problème très souvent rencontré dans le Deep Learning. Il existe de nombreuses techniques qui permettent d'éviter que ce problème :

— **Le Dropout :** Technique qui consiste à la désactivation aléatoire de certains poids à chaque passage d'apprentissage.
— **L'Early Stopping :** Technique qui consiste à l'arrêt prématuré de la phase d'apprentissage.

7 Conclusion

Finalement, ce projet était un véritable défi éthique et technique au coeur de l'actualité sur lequel nous avons beaucoup aimé travailler. L'utilisation d'une boucle adversariale pour débiaiser un CV est pour nous une piste intéressante mais qui n'aboutit pas rapidement à des résultats satisfaisants. En effet, malgré le test de différentes combinaisons des paramètres de nos modèles, nous n'arrivons pas à concilier débiaisement du genre et restitution satisfaisante des informations d'un CV sans faire de sur-apprentissage sur l'ensemble d'apprentissage. Il nous est donc nécessaire d'explorer d'autres pistes, tel la profondeur des réseaux neuronaux, l'équilibrage des données en entrée ou encore l'utilisation d'autres modèles pour représenter le CV à l'entrée de nos modèles.
Nous ne prenons en effet pas compte pour le moment du nombre d'occurrence d'un mot ni de son contexte.

Au delà de tester différents paramètres pour trouver un meilleur modèle, nous pouvons également explorer d'autres pistes tout aussi intéressantes. Nous avons en effet pensé à utiliser un classificateur de compétences au lieu de l'autoencodeur. La boucle adversariale permettrait ainsi d'extraire des compétences d'un CV tout en débiaisant le rapport entre les compétences extraites et le genre du CV.

Dans tous les cas, ce projet nous a vraiment beaucoup appris sur le domaine du Deep Learning, de l'interprétabilité et de façon plus générale d'équité en Machine Learning. Nous allons continuer ce projet dans le cadre du cours de Deep Learning pour explorer différentes possibilités et améliorer nos modèles et leur architecture pour arriver à de meilleurs résultats.

8 Bibliographie

Papiers de recherche :

Towards A Rigorous Science of Interpretable Machine Learning - Finale Doshi-Velez, Been Kim, 2017
Semantics derived automatically from languagecorpora contain human-like biases - Aylin Caliskan, Joanna J. Bryson, Arvind Narayanan, 2017
Learning from Simulated and Unsupervised Images through Adversarial Training - Shrivastava, 2017
The Mythos of Model Interpretability - Zachary C. Lipton, 2017
Matching Jobs and Resumes : a Deep Collaborative Filtering Task - Thomas Schmitt, 2016
Mitigating Unwanted Biases with Adversarial Learning - Brian Hu Zhang, Blake Lemoine, Margaret Mitchell, 2018

Deep Learning MOOCS :

- https ://course.fast.ai/
- https ://classroom.udacity.com/courses/ud730
- *Cours Deep Learning CentraleSupélec* : https ://github.com/hleborgne/TPDL

Documentation technique :

- https ://www.tensorflow.org/
- https ://keras.io/
- https ://medium.com/datadriveninvestor/deep-autoencoder-using-keras-b77cd3e8be95

Taches	Chiffrage	Avancement	Critere
Choix du biais étudié	1		Validation encadrante
Etude Documentaire			
Biais	4		/
Matching	4		/
Interprétabililté	3		/
Choix de la zone d'action pour débiaisement	5		Validation encadrante
Montee en competence en Deep Learning	7		/
Recuperation et comprehension des donnees			
Recuperation des donnees	5		Donnee tierce en cours de recuperation
Exploration du biais dans les données	7		Liste de termes significatifs
Implementation d'une solution de debiaisement			
Acces GPU	2		Colab et fusion (mésocentre)
Creation d'un algo de represensation de CV	12		/
Creation d'un classificateur h/f	8		/
Elimination du biais de l'algo de representation	12		Resultat non significatif pour le classificateur
Evaluation de la perte d'information			
Implementation d'un algorithmme de matching offre/CV	9		/
Choix d'une mesure de perte d'information	12		Validation encadrante
Reflexion sur la correction	9		Determination d'un seuil acceptable
Total	100		