LU2IN013

Recommandation et analyse de sous-titres

Encadré par :

- Nicolas BASKIOTIS
- Vincent GUIGUE

Realisé par :

- Sara MEZIANE
- Mariia BARANOVA
- Mouna AHAMDY

ANNEE UNIVERSITAIRE 2020/2021

Table des matières

1 Introduction

Les algorithmes de recommandation sont omniprésents et rythment discrètement notre navigation sur Internet. Ils sont aujourd'hui indispensables à la presse, aux sites et applications musicales et cinématographiques, aux restaurations. Ces algorithmes améliorent considérablement l'expérience utilisateur car permettent de la personnaliser suivant ses différents goûts et préférences. Ces algorithmes vont plus loin que répondre aux attentes de ce dernier, ils sont même capables de prédire son comportement dans des contextes divers et variés. De plus, ils augmentent considérablement le profit des plateformes qui en font usage. Par exemple, dans le cadre des e-commerces, ils peuvent provoquer de nouveaux achats que les utilisateurs ne considéraient pas préalablement. Ils facilitent également le filtrage des données et permettent de réaliser une analyse beaucoup plus pointue de ces dernières. Cette compétence, pour un moteur générant des millions de résultats par exemple, est indispensable afin d'assurer l'affichage de résultats pertinents satisfaisants les attentes du client.

Pour mettre à terme ce projet, nous avons à notre disposition un ensemble de séries télévisées. Celles-ci sont composées d'un nombre de saisons, qui sont, à leur tour, constituées d'un nombre fini d'épisodes. Ces derniers sont représentés sous format textuel, par leurs sous-titres, et constituent notre champ de travail.

La manipulation des données textuelles n'étant pas une tâche évidente, il nous est donc nécessaire de trouver une représentation simple et efficace de celles-ci. Chaque épisode sera en fait caractérisé par la liste des mots le constituant et représenté sous forme vectorielle, nous pourrions ainsi le qualifier de "sac de mot". De plus, dans un souci de travailler avec des mots pertinents, caractérisant le plus exactement possible nos épisodes et séries, il serait ensuite question de prétraiter ces listes de mots afin de les préparer à l'élaboration de nos algorithmes. La prise en main de nos données textuelles, l'extraction des listes de mots à partir des fichiers mis à disposition, le prétraitement et la présentation de la manipulation de nos données constitue donc la première partie de notre rapport. Les éléments présentés dans ce dernier peuvent être qualifiés comme "outillage", et ce chapitre peut être qualifié comme "coulisses", car nous y présentons tout le travail et les outils permettant de mettre en oeuvre des algorithmes, qui constituent l'objet principal de ce projet.

Les algorithmes de recommandations étant assez complexes, nous choisissons de nous intéresser aux algorithmes de classification - les grands classiques du Machine Learning, pour pouvoir ensuite traiter purement la recommandation des séries. En effet, dans la deuxième partie du rapport, notre objectif sera essentiellement de déterminer l'appartenance d'un épisode inconnu à une certaine série. Ce chapitre, qui est le cœur de notre projet, sera axé autour de trois algorithmes fondamentaux dans l'apprentissage supervisé, à savoir l'algorithme des k plus proches voisins, du perceptron et du perceptron multi classe. Le concept d'apprentissage supervisé ainsi que le fonctionnement de ces algorithmes sera bien évidemment détaillé par la suite. Il est cependant indispensable de mentionner un principe important dans cette étape de classification. Il s'agirait de l'utilisation d'une partie de nos données pour l'apprentissage des algorithmes, et une autre

pour tester les performances de ces derniers. Bien sûr, une partie de ce chapitre sera dédiée à la présentation et l'analyse des résultats obtenus.

Après avoir accompli toutes ces tâches, nous pouvons enfin, dans la troisième partie, parler de la recommandation des séries. Cette dernière est basée sur l'usage des algorithmes vus dans la deuxième partie. Dans ce chapitre, nous verrons donc comment adapter les algorithmes des k plus proches voisins et du perceptron à notre contexte afin de répondre à la problématique principale du projet. Cette partie sera plus théorique que la précédente. En effet, nous imaginerons des exemples d'implémentation de ces algorithmes, et donnerons plusieurs contextes possibles à notre recommandation de séries. Notons cependant qu'il sera difficile d'évaluer nos résultats dans le cadre purement académique de notre projet bien qu'il soit bel et bien possible de les évaluer dans le monde réel. La différence serait la possibilité d'observer la satisfaction de l'utilisateur.

Nous pouvons à présent rentrer dans le vif du sujet.

2 Manipulation de données textuelles

2.1 Introduction

Ce premier chapitre sera consacré à la prise en main des données textuelles dont nous sommes en disposition. En effet, nous allons, dans un premier temps, expliquer la représentation du Data-Set basée sur les mots contenus dans les sous-titres. Nous manipulerons ensuite différents outils de prétraitement de texte et d'analyse statistique afin d'obtenir des données propres et cohérentes utilisables pour la représentation des épisodes. Finalement, nous introduirons les matrices représentatives de la distance entre ces derniers. Celles-ci seront particulièrement utiles dans la conception des algorithmes de classification.

2.2 Représentation des données

Dans un premier temps, nous allons comprendre comment représenter et stocker nos données, afin de conserver uniquement les informations essentielles.

- Nous travaillons avec les sous-titres de nos séries et, comme dit dans l'introduction, nous voudrions associer une liste de mots à chaque épisode, ainsi il nous paraît indispensable de créer le champ lexical de toutes les séries manipulées. On le stockera sous forme de dictionnaire dont la taille sera égale au nombre total des mots différents rencontrés dans tout notre corpus, on la note n.
- Afin de mettre à terme les algorithmes de classification, nous devons trouver une façon de représenter chaque épisode dans un espace. Nous nous plaçons donc dans l'espace vectoriel $\mathbb{R}^n$, où n est la taille du champ lexical, dans lequel chaque épisode sera représenté par un vecteur. Chaque coefficient de ce vecteur sera égal au nombre d'occurrences de chaque mot du dictionnaire dans cet épisode. Ainsi, l'ensemble des épisodes d'une série a son propre champ lexical, le distinguant des autres épisodes des autres séries.
- En apprentissage supervisé, l'ensemble des données manipulées est classiquement nommé Data-Set. Le Data-Set contient toujours deux variables : la target variable Y, qui représente ce que nous souhaitons que la machine prédise, et les features X qui influencent la valeur de Y. Dans notre cas, la matrice X sera la matrice dont les lignes sont les vecteurs-épisodes : elle aura donc autant de lignes que d'épisodes et autant de colonnes que des mots du dictionnaire. Quant à la matrice Y, celle-ci, tout comme la matrice X, contient autant de lignes que le nombre total d'épisodes dans nos séries. Elle contient également trois colonnes : la première représente le numéro d'épisode, la deuxième le numéro de la saison et la troisième indique le numéro de la série.
- Un dernier objet que nous souhaitons définir et qui sera utile dans le troisième chapitre où nous parlerons de recommandations est la matrice agrégée des séries. En effet, à la place de donner la représentation vectorielle d'un épisode, nous donnerons celle d'une série entière : elle contiendra le nombre moyen d'occurrences de chaque mot du vocabulaire dans la série. Les vecteurs obtenus seront ensuite stockés dans une matrice, notée X_a.

épisode	saison	série
0	1	0
24	2	5
5	4	2

Mot / épisode	mot1	mot2	mot3
Ep 1	40	1	15
Ep2	10	0	20
Ep3	5	4	100

Le nombre d'occurences du mot3 dans l'episode 1

FIGURE 1 – Une représentation de la matrice Y (à gauche) et de la matrice X

Une telle représentation vectorielle de nos données (la matrice X) correspond aux objectifs de la classification. En effet, elle permet de distinguer nos épisodes entre eux car elle reflète exactement leur champ lexical. De plus, l'ensemble des épisodes d'une série a également son propre champ lexical qui la distingue bien d'une autre série.

2.3 Prétraitement des données textuelles

Après avoir fixé les objets avec lesquels nous voulons travailler : à savoir le dictionnaire et les matrices X et Y, il serait question, à présent, de les créer.

Pour ce qui est de l'extraction des mots de nos fichiers textuels, nous procédons de la façon suivante. Nous effectuons d'abord une lecture des fichiers, pour entamer ensuite la tokenisation : il s'agit de l'extraction des mots avec l'usage d'un pattern, et notamment celui des Regular Expressions. Cependant, nous remarquons que la liste des mots obtenus à l'issue de cette tokenisation comporte des mots beaucoup trop communs. Ces derniers sont majoritairement des pronoms, des prépositions, des articles et des conjonctions. Une illustration de ce phénomène est donnée par le Word Cloud (a) qui relève des mots les plus fréquents de nos sous-titres.

Ces mots, n'étant propres à aucune série particulière, doivent donc être filtrés et ce filtrage sera effectué grâce à une liste de Stop Words fournie dans la bibliothèque WordCloud. Cette liste est, en effet, constituée des mots anglais les plus courants : pronoms, articles, verbes comme "will, "can"" etc..., ne permettants donc pas de différencier entre les épisodes car ne relèvent pas d'un champ lexical précis. Nous choisirons également de supprimer les mots jugés trop courts, i.e. inférieurs ou égales à 2 lettres, dont l'intérêt pour notre projet est très limité. En régénérant de nouveau un Word Cloud (b), nous obtenons une illustration relativement différente de celle obtenue précédemment.

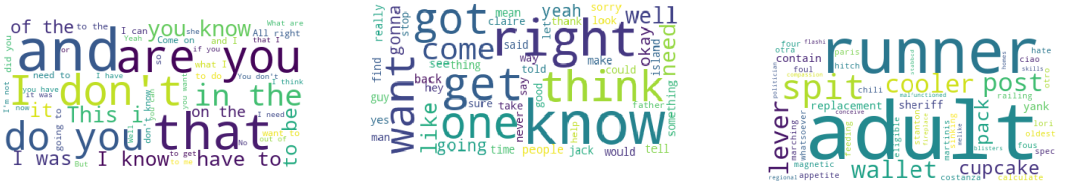
Cependant, le prétraitement des données n'est pas encore terminé. En effet, après les mots communs et les mots relativement courts, d'autres mots se révèlent non intéressants à notre étude car ne nous fournissent aucun renseignement particulier sur nos séries et risqueraient même de fausser nos prochains calculs (en biaisant notamment les calculs de distances entre les divers épisodes). Parmi ces derniers, nous pourrons compter les noms des personnages, des villes etc... (voir le nom „Jack" sur Figure 2 (b)). De plus, nous noterons que la liste des Stop Words fournie n'est pas suffisamment complète : en effet, il existe des mots non intéressants comme "know", "think", "want", ... absents de cette liste.

Tout ceci nous mène alors à l'introduction des concepts de Document Frequency et de Inverse Document Frequency (TF-IDF). En effet, comme leur nom l'indique, ces concepts permettent de déterminer la fréquence d'apparition des mots dans les fichiers. Cette dernière se calcule avec les formules suivantes :

$$
\begin{aligned}
DF(m) &= \frac{N(m)}{N} \\
IDF(m) &= \log_{10} \frac{N}{DF(m)+1} \\
TF(m,d) &= \frac{N(m,d)}{N(mots)} \\
TF-IDF(m,d) &= TF(m,d) * IDF(m)
\end{aligned}
$$

Quelques précisions : N désigne le nombre de fichiers, $N(m)$ désigne le nombre d'occurences du mot m et $N(m,d)$ désigne le nombre d'occurences du mot m dans un fichier d. Finalement $N(mots)$ désigne le nombre de mots dans le fichier.

A noter que l'indice des mots les plus récurrents, et donc moins importants, se rapproche de 0, tandis que celui des mots les plus significatifs se rapproche de 1. Après avoir éliminé 10% des mots les plus récurrents, nous observons l'apparition de mots caractérisant beaucoup mieux les séries étudiées. Ceci est particulièrement visible sur le Word Cloud (c) .



(a) Word cloud avant prétraitement (b) Word cloud avec stop words (c) Word cloud avec TF-IDF

FIGURE 2 – Word clouds

2.4 Matrices de distances/similaritées

Passons à présent à des structures classiques du Machine Learning : il s'agit des matrices dites de distance ou de similarité. En effet, il est important dans le cadre de ce projet de pouvoir observer à quel point deux épisodes, ou deux séries, sont différents l'un de l'autre, afin de pouvoir mettre à terme nos recommandations. Une façon de le visualiser serait de calculer la distance entre les vecteurs-épisodes extraits de la matrice X. A noter qu'il existe plusieurs façons en mathématiques de calculer une distance dans un espace vectoriel, nous avons utilisé pour notre étude les notions de distance euclidienne

et de distance cosine.
La distance euclidienne entre deux vecteurs x_i et x_j en dimension n est calculé ainsi :

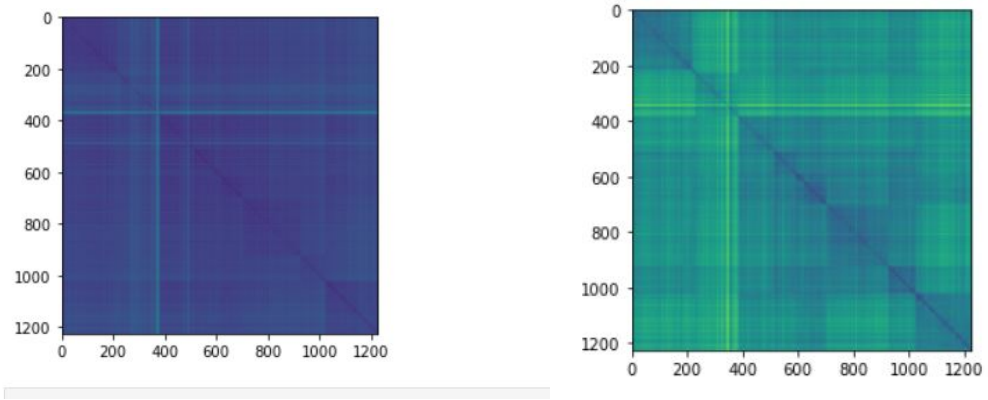
$$d\left(x_i, x_j\right) = \sqrt{\sum_{k=1}^{n} \left(x_{ik} - x_{jk}\right)^2} \tag{1}$$

La distance cosine sert à calculer la similarité de deux vecteurs x_i et x_j à n dimensions en déterminant le cosinus de leur angle.

$$d\left(x_i, x_j\right) = 1 - \frac{\sum_{k=1}^{n} \left(x_{ik}.x_{jk}\right)}{\sqrt{\sum_{k=1}^{n} \left(x_{ik}\right)^2}\sqrt{\sum_{k=1}^{n} \left(x_{jk}\right)^2}} \tag{2}$$

Il est important de souligner que la distance cosine est beaucoup plus intéressante dans le cadre de notre projet. En effet, la matrice X est une matrice sparse, c'est-à-dire une matrice contenant beaucoup de coefficients nuls. Ceci est notamment lié à la présence d'un grand nombre de mots dans le dictionnaire et beaucoup moins de mots différents dans un épisode. Ainsi, nous pouvons observer une ressemblance entre différents épisodes x_i et x_j due à une grande quantité de coefficients nuls. Or, cette ressemblance est totalement sans intérêt. C'est pourquoi nous choisissons d'utiliser la distance cosine plutôt que la distance euclidienne. Nous remarquons également, à partir du numérateur dans la formule (2), que dès qu'une coordonnée de l'un des vecteurs x_i ou x_j est nulle, le mot correspondant à cette coordonnée n'aura aucun impact sur la somme. Ainsi, nous nous intéressons uniquement aux mots dont le nombre d'occurrences est non nul dans les deux épisodes x_i et x_j. Ce sont ces derniers qui influencent la somme et qui, donc, font monter la similarité entre les épisodes.

Nous pouvons finalement construire les matrices des distances euclidiennes et cosines, en voici une représentation construite à partir de 10 séries :



(a) Matrice distance euclidienne

(b) Matrice distance cosine

FIGURE 3 – Matrices des distances entre épisodes paire à paire

(bleu : petites distances, jaune : grandes distances)

Nous pouvons remarquer sur les deux matrices de la Figure 3 la présence d'un trait particulièrement foncé sur les diagonales. En effet, ce trait indique la présence de valeurs nulles. Ceci est tout simplement lié au fait que les coefficients de la diagonale de notre matrice correspondent à la distance d'un épisode à lui-même, qui est nulle. Nous remarquons également, dans les deux matrices, la présence de plusieurs blocs où les distances semblent se rapprocher : ces blocs-là représentent les épisodes d'une même série. On observe dans la matrice des distances euclidiennes deux lignes jaunes, cela signifie qu'il existe un épisode particulièrement éloigné des autres. Après avoir recherché dans nos dossiers, nous avons pu nous apercevoir qu'il s'agissait en fait d'un épisode contenants un grand nombre de mots en espagnol. Finalement, on voit bien que la matrice des distances cosines montre mieux les différences entre les épisodes (plus de traits jaunes), ce qui correspond à l'explication donnée ci-dessus.

La Figure 4 représente un histogramme contenant la distribution des distances entre les épisodes paire à paire. Nous remarquons que le nombre d'épisodes proches les uns des autres est assez élevé, ce sont en grande partie les épisodes appartenant à la même série. Le nombre de paires d'épisodes étant à des grandes distances les uns des autres diminue, ce qui signifie que les séries analysés ont un champ lexical assez similaire.

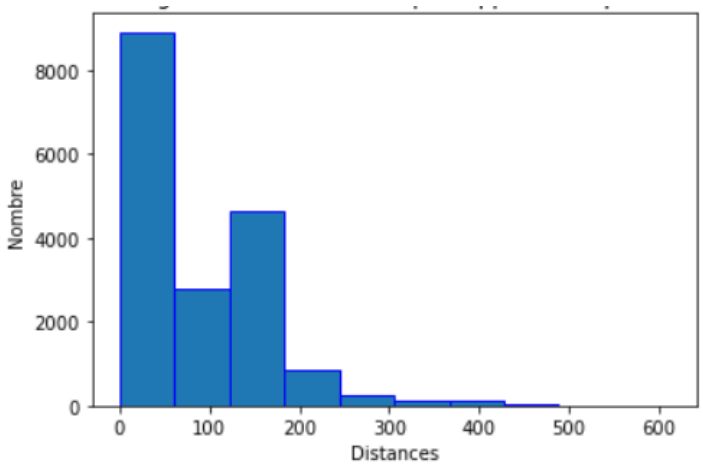


FIGURE 4 – Nombre de couple d'épisodes par distance les séparants

Avec la même méthode nous pourrons construire les matrices de distances entre les séries, à partir de la matrice agrégé X_a.

2.5 Conclusion

Après avoir effectué les opérations de prétraitement nécessaires, nous avons pu créer un champ lexical complet de nos séries, à partir duquel nous avons pu représenter nos données en tant que vecteurs, dans un espace vectoriel. De même, nous nous sommes intéressées à la création des matrices de distances et de similarité, permettant d'avoir une représentation mathématique des différences entre les épisodes. Ces dernières, ainsi que la représentation des épisodes sous forme matricielle permettront une manipulation plus simple des données dans les algorithmes des chapitres suivants.

3 Classification des données

3.1 Introduction

Avant de passer à l'objectif principal de notre projet, à savoir les algorithmes de recommandation, nous allons nous intéresser en détail aux algorithmes de classification. Ces algorithmes étant plus simples à comprendre et à évaluer forment une parfaite base pour les algorithmes de recommandation. En effet, dans le prochain chapitre nous verrons comment adapter ces classiques du Machine Learning à notre problématique.
Ainsi, dans ce chapitre on étudie en détail les 3 algorithmes de classification les plus connus de l'apprentissage supervisé. Mais avant tout, qu'est ce qu'un algorithme d'apprentissage supervisé ? Un tel algorithme prend en entrée des données labellisées d'un ensemble d'apprentissage et permet la classification des données de l'ensemble de test. Dans notre cas, l'objectif est de prédire le numéro de la série à laquelle appartient un épisode donné . En effet, on fournit à notre algorithme des épisodes dont on connaît le numéro de série, à partir desquelles il peut "apprendre", pour ensuite lui donner un numéro d'épisode que l'on voudra classifier.
Nous allons d'abord présenter le protocole expérimental, en particulier parler de la subdivision de notre corpus en une partie dite d'apprentissage ainsi qu'une partie test. Ensuite, nous allons détailler les principes de fonctionnement des algorithmes kPPV (les k plus proches voisins), du perceptron et du perceptron multi classe. On s'intéressera également à leur efficacité, leur coût et on testera l'exactitude de leurs prédictions.

3.2 Protocol expérimental

Comme dit précédemment, notre objectif est de créer un algorithme capable de déterminer à quelle série peut appartenir un certain épisode en connaissant sa position dans l'espace vectoriel. Les algorithmes d'apprentissage supervisé sont donc divisés en deux parties, l'apprentissage et la classification. Un procédé classique dans le Machine Learning consiste à diviser l'ensemble de nos données, ici les matrices X et Y, en un ensemble d'apprentissage, "train", permettant à l'algorithme d'apprendre, et en une partie d'évaluation, "test", que nous voudrions classifier. Il est important de faire une telle division pour créer un modèle performant et obtenir des résultats de tests fiables. En effet, apprendre et tester sur le même ensemble de données ne permettrait pas de généraliser le modèle et pourrait devenir une source d'erreurs. Ce phénomène est illustré dans la Figure 5. Il existe plusieurs façons de subdiviser un corpus, mais afin de les mettre à terme, nous devons respecter quelques principes :

- Le sous-corpus d'apprentissage doit être constitué de 70%-90% de l'ensemble original afin d'obtenir un modèle fiable, les 10%-30% restants doivent êtres gardés pour les tests.
- Les sous-corpus doivent être représentatifs de l'ensemble global des données.
- Pendant cette séparation, nous prenons soin de mélanger les données.

Une méthode commune pour réaliser cette division est la validation croisée. Elle consiste à diviser l'ensemble de données en k parties, nommées plis. A la première ité-

ration, le premier pli permet de tester le modèle et les autres servent à le former. A l'itération suivante, le deuxième pli est utilisé pour le test et les autres plis pour l'apprentissage. Et ainsi de suite, jusqu'à ce que chaque pli soit utilisé au moins une fois dans le test. Cette méthode garantit qu'il n'y ait pas de lien entre le score du modèle et la technique adoptée pour le choix de l'ensemble des données de test/d'entraînement.

FIGURE 5 – Division du corpus

3.3 Les k plus proches voisins (kPPV)

Le premier algorithme de classification que nous avons eu l'opportunité d'étudier est aujourd'hui compté parmi l'un des algorithmes les plus simples du Machine Learning. Il s'agit en effet de l'algorithme des k Plus Proches Voisins, aussi nommé kPPV -en français- ou kNN –en anglais-.

Intéressons-nous à son fonctionnement. kPPV repose majoritairement sur le calcul de la similarité cosine, ou la distance euclidienne, de tous les épisodes composant notre fichier. Supposons que nous traitons un épisode x. Les étapes que devrait suivre notre algorithme sont les suivantes :

1- Nous sélectionnons d'abord les k épisodes les moins distants de x dans la matrice de distance M.

2- Nous attribuons à chaque épisode sélectionné la série à laquelle il appartient en ayant recours à la matrice d'épisodes Y.

3- Nous choisissons la série la plus occurrente parmi les séries attribuées à l'étape 2). C'est alors à cette série qu'appartient notre épisode x.

Dans un second temps, après avoir compris son objectif et son fonctionnement, il est intéressant de tester la performance et l'exactitude de cet algorithme. Pour calculer sa performance, nous avons choisi de porter l'étude sur les épisodes composant la matrice Y_test. Une fois, que l'algorithme kPPV a attribué à chaque épisode x de Y_test une série béta, il serait question de savoir si béta est réellement la série contenant x ou si notre algorithme s'est trompé.

Pour se donner une idée de la performance de l'algorithme des kPPV, nous calculons le pourcentage de réussite de notre enquête, i.e, le pourcentage du nombre de fois où l'on retrouve que bêta est bien la série à laquelle appartient x. Nous noterons que ce pourcentage diffère en fonction du nombre k choisis. Contrairement à ce que nous aurions

pu penser, le pourcentage de réussite ne grandit pas particulièrement avec l'augmentation du nombre k. Il est cependant plus intéressant de travailler avec un k impair, car il contraint l'algorithme à choisir une seule série. Si nous travaillons avec un nombre pair, nos résultats seraient moins cohérents car nous pourrions avoir un cas de figure où nous aurions 50% de chance que notre épisode appartienne à une série a, et 50% de chances qu'il appartienne à une autre série b. Auquel cas notre algorithme selectionnera aléatoirement une parmi les 2. Le résultat serait donc moins fiable.

Voici quelques exemples de suggestions de séries obtenus pour des épisodes différents :

	Série prédit	Série
0	24	24
1	Heroes	Heroes
2	Heroes	Heroes
3	Doctor_Who	Doctor_Who
4	24	24
..	...	...
302	Supernatural	Supernatural
303	Prison_Break	Lost
304	Heroes	Heroes
305	Doctor_Who	Doctor_Who
306	Prison_Break	Prison_Break

FIGURE 6 – Jeu de test avec dix séries

Voici quelques pourcentages de réussite obtenus pour des choix de k différents :

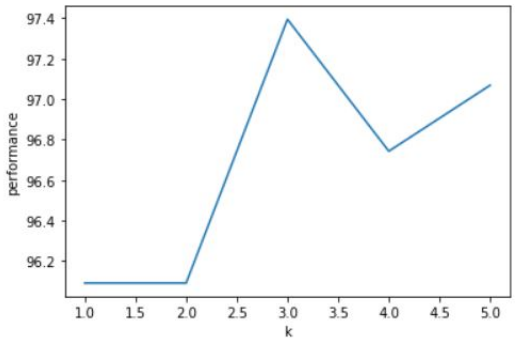


FIGURE 7 – Jeu de test avec dix séries

3.4 Le Perceptron

Le deuxième algorithme de classification que nous abordons dans le cadre de notre projet est le perceptron - le premier algorithme de Machine Learning qui se trouve à la base des réseaux de neurones. Dans cette partie nous allons d'abord expliquer le principe de fonctionnement de cet algorithme, pour après montrer son application à nos données.

Principe de l'algorithme

Le perceptron est un algorithme d'apprentissage supervisé binaire. Son objectif est de séparer linéairement deux classes de points. Ainsi, dans le cadre du perceptron, nous allons travailler avec des épisodes de deux séries différentes uniquement.

Dans un premier temps, la partie d'apprentissage du perceptron consistera à trouver un hyperplan permettant de séparer linéairement au mieux les épisodes des deux séries (on rappelle que chaque épisode est un vecteur dans l'espace R^n).
Le document suivant est une représentation fictive (ayant pour seul but d'illustrer l'explication) des épisodes dans un espace de dimension 2 (comme si le dictionnaire contenait 2 mots seulement). Les épisodes de la première série sont les points oranges et la deuxième série les points bleus. Le séparateur linéaire est dans ce cas une simple droite.

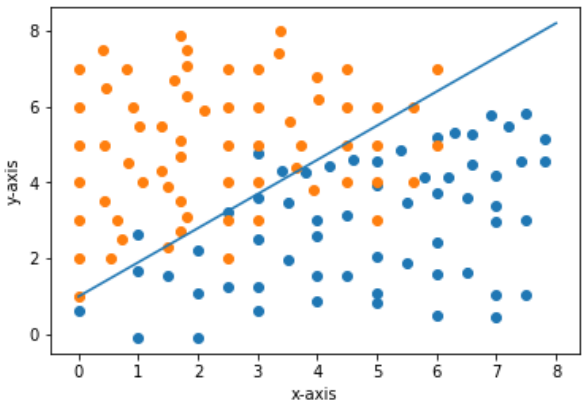


FIGURE 8 – Representation d'un perceptron

Comme le perceptron prend des données labellisées, on associe le label 1 aux épisodes qui se trouvent dans la première série et -1 pour les épisodes de la deuxième série. On note "y" le label de chaque épisode. Plus précisément, 1 : correspond aux épisodes au-dessus de l'hyperplan , et -1 en-dessous.
Le vecteur directeur de l'hyperplan que l'on cherche à déterminer, est nommé le classifieur et il sera renvoyé par le perceptron. On le note w par la suite.

Une partie importante et très spécifique à cet algorithme est le fait que à priori on sait pas à quel moment on doit arrêter de modifier w. Pour résoudre cette problématique on choisit de calculer le coût de classification à chaque tour, et on décide du moment d'arrêt en fonction de la moyenne des coûts. Les formules de calcul du coût sont données ci-dessous :

$$l\left(f_w(x^i, y^i)\right) = max(0, -f_w(x^i) \times y^i) \tag{3}$$

$$L\left(f_w, (X, Y)\right)) = \frac{1}{N} \times \sum l\left(f_w(x^i, y^i)\right) \tag{4}$$

- On note le produit scalaire $f_w(x) =< w, x >$ calculé pour le $i$ème épisode.
- N est le nombre total d'épisodes considérés.

Déroulement de l'apprentissage du perceptron

Voici un pseudo-algorithme du perceptron :

- On choisit aléatoirement les épisodes dans l'ensemble d'apprentissage X_train et on initialise le classifieur w aléatoirement.
- Soit x un épisode de X_train choisi. On va maintenant vérifier s'il est bien positionné par rapport à l'hyperplan.
  Si $f_w(x).y >= 0$, alors l'épisode x est bien classé et on ne modifie pas w. Sinon, on met à jour w : $w = w + \epsilon.y.x$. On appelle epsilon le pas d'apprentissage, un paramètre que l'on fixe.
- On réitère ce processus sur tous les épisodes de la matrice X_train pris dans un ordre aléatoire. Cet ordre aléatoire de choisir des épisodes est en fait une permutation, que l'on appelle une époque. On réitérera ce même processus avec plusieurs autres époques différentes dont le nombre reste à déterminer.

Classifiaction, campagne d'expériences et résultats

Afin de déterminer le nombre d'époques nous traçons le gaphique représenté sur la Figure 9. Nous observons qu'il suffit d'une seule époque pour que la fonction coût arrive à 0. Ceci est lié au fait que les données étudiées, les deux séries, ont des champs lexicaux très différents. De plus, les épisodes sont facilement séparables par le classifieur. Nous allons quand même implémenter notre code sur au moins 10 époques différentes, ce qui permettra de couvrir des cas de figure plus compliqués.

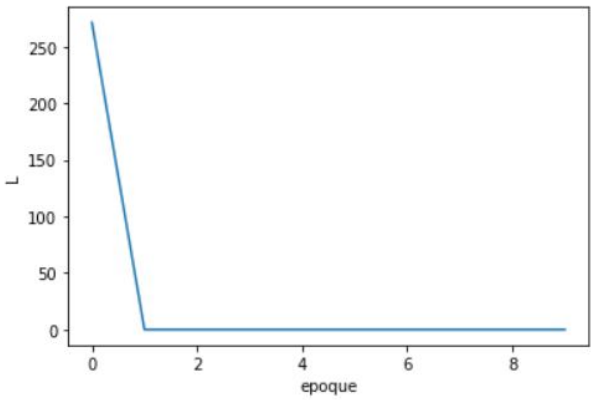


FIGURE 9 – Fonction coût L en fonction des époques

Après avoir calculé le classifieur, il est nécessaire de le tester sur l'ensemble des données tests. La Figure 10 représente les résultats des classifications des données tests de deux séries : "Lost" et "Heroes". On remarque que dans notre cas l'algorithme effectue

un classement sans faille. Cette efficacité est liée, comme dit précédemment, à la différence notable des champs lexicaux des deux séries. Ceci est également confirmé par le calcul de performance qui tournait souvent dans nos expériences autour de 95-100% .

	Série prédit	Série
0	Lost	Lost
1	Lost	Lost
2	Lost	Lost
3	Lost	Lost
4	Lost	Lost
5	Lost	Lost
6	Lost	Lost
7	Heroes	Heroes
8	Heroes	Heroes
9	Lost	Lost
10	Lost	Lost

FIGURE 10 – Jeu de tests classification séries "Lost" et "Heroes"

En fonction des coordonnées du classifieur nous pouvons observer les mots qui ont le plus grand et le plus petit impact sur la classification. La Figure 11 présente les listes des 40 mots les plus et moins influents dans la classification des séries “Suits” et “Boston legal”, qui sont des séries judiciaires. Nous remarquons que parmi les mots les plus influents, des mots relèvent de ce même thème judicier, parmi ces derniers nous pouvons compter : "law" , "shareHolders", "partners"... A l’opposé, nous trouvons, dans la liste des mots les moins influents, des mots qui n’ont pas de grande importance et qui ne relèvent point du champ lexical de la série.

```
 les mots les plus influents sont :
['place', 'earn', 'signed', 'changed', 'heat', 'picked', 'guess', 'best', 'southern', 'me
mo', 'minutes', 'zane', 'four', 'law', 'walking', 'like', 'department', 'color', 'crap',
'streamline', 'come', 'puberty', 'taller', 'want', 'middle', 'normally', 'summer', 'rat
e', 'rachel', 'asks', 'around', 'whipping', 'handled', 'meal', 'sainz', 'real', 'nights',
'screwed', 'happening', 'check']
****************************************
 les mots les moins influents sont :
['neutralized', 'straits', 'reattach', 'bootlegging', 'pope', 'diocese', 'blessings', 'un
lock', 'italy', 'imported', 'parishioners', 'abduction', 'pertaining', 'receipt', 'exchan
ges', 'canon', 'unfortunely', 'beadle', 'productively', 'robb', 'reprise', 'fonda', 'diso
bedience', 'medicaid', 'plunged', 'hijacking', 'hijacker', 'hosting', 'gracie', 'paramili
tary', 'alias', 'revere', 'gigantic', 'drivel', 'entertained', 'hanoi', 'hunts', 'barbare
lla', 'lest', 'jane']
```

FIGURE 11 – Les mots les plus influents et les moins influents dans "Suits" et "Boston Legal"

3.5 Le Perceptron Multiclasse

Le perceptron, étant un algorithme binaire, permet de classifier uniquement les épisodes de deux séries différentes. Or, nos données contiennent des épisodes d’une centaine de séries, que nous souhaiterions tous classifier. Pour faire cela, nous avons donc opté pour

l'algorithme de multi-classe qui exploite les propriétés du perceptron, tout en étant n-aire.

Principe de l'algorithme

- Dérouler un multiclasse revient à dérouler autant de perceptrons que de séries dans l'ensemble d'entraînement, c'est -à -dire, changer les directions de plusieurs vecteurs directeurs de plusieurs hyperplans jusqu'à ce que l'on parvienne à séparer uniformément les épisodes dans chacun.
- Pour la labellisation de nos séries, autrefois, nous attribuions -1 à une série, et 1 à l'autre, dans le but de les séparer. Nous procéderons de façon similaire ici. Plus précisément, pour chaque perceptron, nous choisirons une série qui sera labellisée 1 et les autres séries seront labellisées -1 : c'est le concept d' "une série contre tous".
- Chaque perceptron renvoie un classifieur w_i et chacun de ces classifieurs constitue une ligne d'une nouvelle matrice W.

Classifiaction, campagne d'expériences et résultats

Après la phase d'apprentissage de l'algorithme, nous commençons à classifier les séries avec l'ensemble de tests. Cela se fera par un produit scalaire de chaque épisode avec chaque classifieur, et le classifieur le plus proche d'un épisode donné nous indiquera son appartenance. Ainsi nous obtiendrons, au final, la classification de tous les épisodes de l'ensemble de test.

Après avoir trouvé les prédictions des épisodes, il ne reste plus qu'à les comparer aux valeurs réelles, et de calculer ainsi le taux de réussite de notre algorithme. Dans notre cas, le taux est relativement élevé, aux alentours de 80-100%, car le champ lexical de chaque série la distingue facilement des autres, ce qui rend la classification rapide et efficace.

Pour identifier les bonnes et les mauvaises classifications nous avons fait le choix de former la matrice de confusion dans laquelle chaque case représente le nombre d'épisodes classés dans les différentes séries.

La Figure 12 fournit un exemple de classification à partir de 10 séries.

En première ligne, par exemple, nous avons 44 épisodes de la première série qui ont été classifiés comme des épisodes de cette même série, ce qui est bien sûr correct. En revanche, un épisode de cette dernière a été classifié dans la cinquième série. Ceci s'explique par le fait que les mots employés dans cet épisode se rapprochent plus du champ lexical de la série numéro 5 que du champ lexical de la série numéro 1. Globalement nous remarquons que la matrice ressemble beaucoup à une matrice diagonale, et ceci est lié au fait que la majorité des épisodes de l'ensemble de test a bien été classifié.

```
 performance du multiclasse :
99.3485342019544
*****************************************
la matrice de confusion est :
[[44.  0.  0.  0.  1.  0.  0.  0.  0.  0.]
 [ 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0. 37.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0. 40.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0. 27.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  7.  0.  0.  0.  1.]
 [ 0.  0.  0.  0.  0.  0. 17.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0. 55.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0. 23.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0. 54.]]
```

FIGURE 12 – Matrice de confusion sur 10 séries

3.6 Conclusion

Après avoir assimilé les concepts fondamentaux des algorithmes d'apprentissage supervisé, nous avons réussi à les appliquer aux modèles les plus connus du Machine Learning afin de pouvoir classifier nos épisodes dans des séries. Nous avions souvent réussi à obtenir des résultats assez cohérents, la performance s'élevant jusqu'à 90%-100% dans la plupart des cas.

4 Recommandation des séries télévisées

4.1 Introduction

Le troisième et dernier chapitre de notre rapport traitera de l'objectif principal du projet : la recommandation des séries télévisées. Il existe plusieurs types d'algorithmes de recommandations que nous allons donc présenter. Une première approche, nommée filtrage collaboratif, consiste à recommander des séries à un utilisateur en se basant sur les notes données à ces séries par des utilisateurs ayant des goûts similaires à ce dernier. Il existe également une deuxième approche, à laquelle nous allons nous intéresser davantage dans le cadre du projet. Il s'agit, en effet, des algorithmes de recommandation basés sur le contenu : "content-based algorithmes". Ce cas de figure est très différent du précédent car il prend en compte uniquement le contenu des séries visionnées et ne repose pas sur les autres utilisateurs ou sur leurs avis. Malheureusement, nous ne pourrons pas conclure sur la performance de ces algorithmes ou sur la véracité du résultat renvoyé dans le cadre de notre projet. Nous aurions, en effet, besoin de connaître le comportement de l'utilisateur face aux séries recommandées, ce qui est uniquement possible dans le monde réel. Par exemple, si ce dernier visionne la série recommandée pendant un temps plus long qu'une autre, nous pourrons juger sa préférence de cette dernière, chose impossible dans notre cadre purement académique.

4.2 Algorithme de recommandation basé sur le Perceptron : Filtrage et Content Based

Pour implémenter un algorithme de recommandation dit avec filtrage, nous nous basons sur l'algorithme du Perceptron dont on a détaillé le fonctionnement dans le chapitre précédent.

Rappelons que le perceptron prend en entrée des données labellisées et sépare linéairement deux types de données. Dans ce contexte de recommandation, nous manipulons toujours des séries. Cependant, ce qui diffère de l'ancienne implémentation est le fait de séparer celles-ci en se basant sur les préférences des utilisateurs. Plus précisément nous allons travailler avec un "perceptron utilisateur". Ainsi, l'ensemble d'apprentissage sera l'ensemble de séries que l'utilisateur a visionné. Son avis sur ces dernières nous permettra de déterminer le label : 1 correspondra aux séries qu'il a aimé, -1 à celles qu'il a le moins appréciées. Ainsi notre ensemble de tests contiendra les séries que nous souhaitons potentiellement recommander.

Nous commençons donc par l'apprentissage effectué par le perceptron se basant sur la liste de séries visionnées par l'utilisateur. Ce dernier retourne donc le classifieur . Il serait à présent question de sélectionner parmi les séries non visionnées celles à recommander. Pour faire ce choix, nous choisirons simplement les séries les plus proches de la partie supérieure de l'hyperplan. En effet, il s'agirait des séries se rapprochant le plus des contenus des séries qu'il a aimé. Afin de les connaître, nous utiliserons leurs représentations vectorielles et le vecteur classifieur. Mathématiquement parlant, le produit scalaire entre ces deux vecteurs définit le score attribué à chaque série : plus ce score est grand, plus le

contenu de celle-ci intéressera potentiellement l'utilisateur. Finalement, nous obtiendrons une liste de séries à recommander.

Pour avoir une idée sur la performance d'un tel système, nous choisissons un utilisateur A, qui a une préférence pour les séries à thème médical, et qui ne veut pas visionner des séries judiciaires. Ainsi, on crée notre ensemble d'apprentissage et de test grâce aux préférences qu'on a , et on exécute l'algorithme décrit précédemment. A présent nous pourrons regarder les résultats obtenus :

```
Pour un utilisateur qui a aimé les séries médicales Grey s Anatomy , New amesterdam et Doctor s Diary
, et qui a moins aimé les séries à thème judiciaire Boston legal , Suits et Damages , on recommande :
Scrubs et The Knick
```

FIGURE 13 – Recommandation de 2 séries pour l'utilisateur A

Notons que les deux séries recommandées, "Scrubs" et "The knick", sont des séries médicales. Ainsi on obtient bien un résultat cohérent et répondant aux préférences de l'utilisateur A.

4.3 Algorithme de recommandation basé sur l'algorithme kPPV : Content Based

Intéressons-nous dans un second temps à un exemple d'algorithme basé uniquement sur le contenu des séries. Pour ce faire, nous utiliserons une approche que nous avons évoquée tout au long du rapport : l'algorithme des k plus proches voisins.

Seulement notre approche diffère de celle évoquée dans le chapitre précédent. Premièrement, il n'est plus dans notre intérêt de travailler avec des données séparées en ensemble d'apprentissage et de test puisque notre objectif n'est plus de déterminer l'appartenance d'un épisode à une série. Pour faciliter la tâche, nous utiliserons uniquement la matrice agrégée de distances M_a. A partir de cette dernière nous sélectionnons les k séries correspondantes aux plus petites distances de la série traitée. Ce sont ces dernières que nous renverrons.

A noter que, par exemple, si nous souhaitons recommander 5 séries à l'utilisateur, nous choisirons un $k = 5$.

Bien qu'il soit très difficile de conclure sur la véracité des résultats obtenus, nous avons choisis de tester cet algorithme sur un ensemble de 10 séries dont la moitié sont des séries judiciaires et l'autre moitié sont des séries médicales (cf. Figure 16).

On s'attend à ce qu'à partir d'une série judiciaire par exemple, l'algorithme nous recommande beaucoup plus de séries du même genre, si pas uniquement ces dernières. Nous remarquons qu'en basant ainsi l'étude sur la série judiciaire "Suits", et en fixant k =3, la recommandation effectuée est tout à fait correcte et assez valide car constituée uniquement de séries du même genre. La figure 14 illustre le résultat obtenu.

```
Pour la série Suits:  on recommande les séries
['Damages', 'Boston_Legal', 'How_To_Get_Away_with_Murder']
```

FIGURE 14 – Recommandation de 3 séries pour la série "Suits"

Cependant, il est important de noter que le pourcentage des mots filtrés dans l'Inverse Document Frequency influence énormément le résultat. Dans des séries comme "Doctor's Diary", filtrer plus de 0.2% des mots les plus occurrents change beaucoup la recommandation car, bien que nous soyons dans un cadre médical, il y a également beaucoup de drames et beaucoup de conversations qui tournent autour de sujets assez éloignés de ce dernier. Ainsi, ce filtrage supprime en réalité la majorité des mots relevant du cadre médical, menant l'algorithme à recommander des séries d'autres genres. Le résultat obtenu n'est donc pas comparable à celui de "Suits" dans lequel nous avons uniquement supprimé les 200 mots les plus fréquents. Une illustration de ce phénomène, en se fixant $k = 5$, se trouve sur la figure 15.

```
Pour la série Doctor_s_Diary:  on recommande les séries
['Damages', 'Scrubs', 'Boston_Legal', 'Grey_s_Anatomy', 'New_Amsterdam']
```

FIGURE 15 – Recommandation de 5 séries pour la série "Doctor's Diary"

Voici un tableau indiquant les genres des séries manipulées dans les tests précédents :

Séries médicales	**Séries judiciaires**
Grey's anatomy	Suits
New Amstardam	How to get away with a murder
Scrubs	Damages
Doctor's Diary	Boston Legal
The Knick	Better Call Saul

FIGURE 16 – Répartition des séries par genre

4.4 Conclusion

Après avoir expliqué les différents cas de figures dans lesquels nous avons choisis de traiter cette question de recommandation, nous avons réussi à réadapter les algorithmes de classification afin de mener à terme notre projet et fournir des recommandations de séries répondant à des contextes variés. Les résultats obtenus sont assez cohérents et semblent valides, bien qu'ils puissent être améliorés en utilisant une approche beaucoup plus approfondies et des outils beaucoup plus pointus.

5 Conclusion

Nous arrivons finalement au bout de notre projet en ayant réussi à recommander des séries pouvant intéresser notre utilisateur. Suite à ce travail, nous avons réalisé, que bien que les algorithmes de recommandation soient d'une très grande utilité, ils nécessitent l'usage de notions assez complexes pour être concrétisés. Nous sommes passées d'abord par la préparation d'objets mathématiques tels que les vecteurs représentants les épisodes et séries, les matrices de distances, de similarité etc. Ensuite, afin de manipuler des données suffisamment précises et cohérentes, nous sommes passées à l'étape de filtrage et d'analyse approfondie. Tout ceci nous permet d'effectuer, avec succès, la classification de nos données. Cette dernière pourra ensuite être réutilisée dans différents contextes de recommandation de séries afin de répondre à notre problématique initiale. Pour conclure, ce projet est une réelle fusion de concepts informatiques et mathématiques, les uns complétant les autres, permettant de répondre à une problématique fondamentale du domaine cinématographique moderne. Cependant, il reste compliqué dans notre cadre académique, d'évaluer le réel impact de notre travail.

Bibliographie

[1] Algorithme n°5 -comprendre la méthode des "k-plus proches voisins". **lien** :https://blog.ysance.com/algorithme-n5-comprendre-la-methode-des-k-plus-proches-voisins-en-5-min, consulté le 18/05/21.

[2] Analyzing documents with tf-idf , programming historian. **lien** :https://programminghistorian.org/en/lessons/analyzing-documents-with-tfidf, consulté le 11/05/21.

[3] Code projet lu2in013. **lien** :https://github.com/MachaBar/PROJET-LU2IN013/tree/main.

[4] Ensembles d'apprentissage et d'évaluation : division des données. **lien** :https://developers.google.com/machine-learning/crash-course/training-and-test-sets/splitting-data?hl=fr, consulté le 27/03/21.

[5] Focus : Le réseau de neurones artificiels ou perceptron multicouche-pensée artificielle. **lien** :https://penseeartificielle.fr/focus-reseau-neurones-artificiels-perceptron-multicouche/, consulté le 20/05/21.

[6] Machine learning : l'algorithme des k plus proches voisins. **lien** :https://moncoachdata.com/blog/algorithme-des-k-plus-proches-voisins/, consulté le 15/06/21.

[7] Numpy and scipy documentation. **lien** :https://docs.scipy.org/doc/, consulté le 24/04/21.

[8] Quels sont les principaux algorithmes de recommandation ? **lien** :https://www.mediego.com/fr/blog/principaux-algorithmes-de-recommandation/, consulté le 13/06/21.

[9] Régression linéaire python -machine learnia. **lien** :https://machinelearnia.com/regression-lineaire-python/, consulté le 15/05/21.

[10] Scipy library. **lien** :https://www.scipy.org/scipylib/index.html, consulté le 24/04/21.

[11] Tutorialrecommender systems in python. **lien** :https://www.datacamp.com/community/tutorials/recommender-systems-python, consulté le 12/06/21.

[12] Tutoriel numpy. **lien** :http://webia.lip6.fr/~mapsi/pmwiki.php?n=Main.TutoPythonJN, consulté le 30/02/21.

[13] What is tf-idf ? **lien** :https://monkeylearn.com/blog/what-is-tf-idf/, consulté le 11/05/21.