Compétition 2 IFT 3395-6390

Rapport

Équipe : print("coucou")
Gabin MAZUÉ, Thomas RIVES, Hamed Nazim MAMACHE
20244318, 20244319, 20244317

Décembre 2022

Table des matières

1 Introduction

Dans le cadre du cours IFT 3395-6390 nous devions participer à une compétition sur la classification de textes. L'objectif est d'implémenter et de tester différents modèles d'apprentissage machine pour classer ces textes. Les données d'entrainement et de test sont des tweets présentés sous la forme de string contenus dans un dataframe. Les classes de ces tweets sont également des strings représentant la positivité de ceux-ci. En effet, si un tweet a une connotation négative alors le label de ce tweet sera "negative". Il existe trois types de labels, "positive", "negative" et "neutral". Pour cette compétiton nous avons accès à un jeu d'entrainement de 1 030 487 données et un jeu de test de 560 175 données.

index	text
0	Anyway Im getting of for a while
1	My red, Apache isn't feelin too well this morn...
2	@danyelljoy you should be its great. friday w...
3	its 11:30pm and i dont wanna sleep; so i debat...
4	Why does twitter eat my DM's? Not happy

FIGURE 1 – Exemple de tweets du jeu d'entrainement

Pour classer ces images, nous allons commencer par implémenter un modèle de classifieur de Bayes naïf. Nous allons ensuite utiliser un modèle SVM. Nous verrons que les performances de ces deux modèles ne sont pas satisfaisantes, ce qui va nous pousser à passer sur des modèles plus robustes tels que MLP, le modèle GRU et le modèle LSTM.

2 Feature Design and Data Cleaning

Avant de commencer à implémenter les modèles et de les entrainer, il est important d'analyser les données et de voir si celles-ci peuvent être transformées pour essayer de réduire la compléxité ou bien mettre en évidence des éléments particuliers qui aideront le modèle à faire de meilleures prédictions.

On commence alors par regarder le pourcentage d'apparition de chaque classe pour voir si le jeu de donnée est balancé ou débalancé :

```
fréquence de positif :  50.11591606686936  %
fréquence de négatifs :  49.875932447473865  %
fréquence de neutres :  0.008151485656781697  %
```

FIGURE 2 – Fréquence d'appartition de chaque classe

Ici, on voit très clairement que notre jeu de données est débalancé et que les textes neutres sont sous représentés.

2.1 Encodage BOW

En parallèle de ce problème, nous devions trouver une manière d'encoder les données pour qu'elles soient lisibles et interprétables par un modèle d'apprentissage. Pour ce faire nous avons commencé par implémenter la méthode d'encodage BOW (Bag Of Words). Le principe est donc de repérer tous les mots de notre ensemble de texte, d'en faire un vocabulaire et de représenter les tweets sous forme de vecteurs (généralement creux) possédant aux indices des mots présents dans celui-ci l'occurence de ces derniers

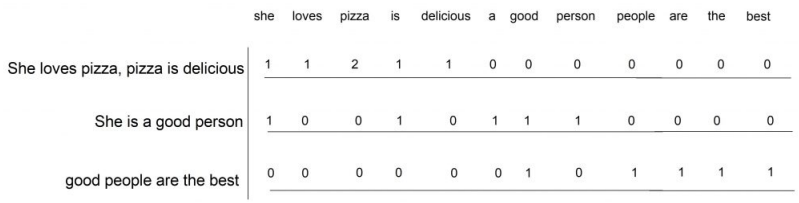

FIGURE 3 – Exemple de représentation BOW

Cette méthode présente un problème majeur qui est celui de la mémoire. En effet même si certains algorithmes d'apprentissage machine peuvent gérer les matrices creuses, la plupart n'en sont pas capables et donc si la taille du vocabulaire est trop grande et que le nombre d'échantillons d'entrainement aussi on se retrouve avec des données ne pouvant même pas être stockées dans la RAM. Nous allons utiliser cette méthode d'encodage pour le classifieur de Bayes naïf, pour le SVM ainsi que pour le MLP.

2.2 Encodage par plongement

Pour pouvoir utiliser des modèles plus robustes et avoir de meilleur résultats, il nous fallait donc parer tous les problèmes de l'encodage précédent. Nous avons donc fait le choix de partir sur de l'encodage par plongement qui consite à créer un espace vectoriel des mots de notre vocabulaire et donc des phrases.

Malgré le fait que cet encodage nous permet de garder le sens des mots et leur position dans la phrase ainsi que de diminuer la dimension et donc le temps de calcul, cette méthode n'est pas parfaite. En effet, il ne faut pas oublier que nous avons à faire à des tweets, et donc nous faisons face au problème de fréquences des mots. En effet, la majorité des mots ne sont utilisés qu'une fois du fait que beaucoup d'individus font des fautes d'orthographe, ou qu'ils utilisent plusieurs fois une même lettre pour donner de l'intensité à un mot, ou encore qu'ils utilisent des caractères spéciaux, des majuscules, des url, des tag à d'autres personnes, etc . Les vecteurs possèdent alors des indices uniques et les modèles ne peuvent donc pas trouver un sens à la représentation. C'est pourquoi nous devons faire appel à la normalisation des données et aux méthodes de rétrécissement du vocabulaire.

2.3 Normalisation

Comme évoqué précédemment, on retrouve dans nos données plusieurs ortographes pour un même mot, de part l'utilisation des majuscules, des caractères spéciaux, des fautes d'ortographe, etc. Nous allons donc mettre toutes les données en minuscule, retirer les espaces en trop et les tabulations, transformer tout les url par la chaine de caractère "url", changer tout les noms d'utilisateur (ex : "@danyelljoy") par la chaine "user" et faire en sorte que toutes les lettres qui se répètent trop (ex : "loooollll") soient remplacées par le mot d'origine (ex : "lol").

2.4 Mots Vides

Il est important de rappeler que les tweets utilisés sont uniquement en anglais et que la langue anglaise, comme la langue française et tant d'autres comprennent des mots n'ayant pas vraiment de sens à part entière, mais qui sont là pour structurer les phrases, genrer ou pluraliser. C'est le cas notamment des déterminants. Notre but ici va être de retirer tout ces mots pour essayer de réduire la taille de notre vocabulaire, mais également de mettre en avant la connotation positive, négative ou neutre du texte.

2.5 Lemmatisation

Le but de cette méthode est de réduire un mot à sa racine et plus précisement à sa connotation de base. En effet, les mots peuvent généralement être expliqués/vulgarisés par d'autre mots beaucoup plus simple. Nous cherchons à simplifier au maximum, pour réduire la taille de notre vocabulaire et donner un sens commun à différents mots. Pour ce faire, la lemmatisation permet d'obtenir les dérivations des mots. Par exemple, le mot "merveilleux" est un dérivé du mot "bon". On remplace donc tout les mots par des synomymes simplifiés.

2.6 Stemming

Le stemming ressemble fortement à la lemmatisation. En effet, le but est également de réduire le mot à sa racine. La différence se fait dans la façon de réduire. Nous avons vu que la lemmatisation change l'entièreté du mot par un synonime plus simple. Le stemming quant à lui se contente de retirer les suffixes des mots :

Form	Suffix	Stem
studies	-es	studi
studying	-ing	study
niñas	-as	niñ
niñez	-ez	niñ

FIGURE 4 – Exemple de stemming

3 Algorithmes

3.1 Classifieur de Bayes Naïf

Pour le premier modèle, il nous a été demander d'implémenter un classifieur de Bayes Naïf en utilisant l'encodage BOW. Pour ce faire nous avons utilisé la bibliothèque Scikit Learn et plus précisement les modèles de Bayes Multinomiales et Gaussiens. Suite à nos tests, nous avons préféré rester sur le modèle multinomiale car capable de gérer l'entrainement sur de matrices creuses (encodage BOW) et donnant de meilleurs résultats.

3.2 SVM

Pour cette deuxième méthode, il nous a été demandé de traiter un SVM en utilisant un kernel pour les chaînes de caractères. Nous avons une nouvelle fois utilisé Scikit Learn et avons opté pour l'utilisation d'un SVC. Pour ce qui est du kernel, nous avons utilisé le kernel pour chaîne de caractères proposé par la bibliothèque stringkernels.

3.3 MLP

Après avoir implémenté le modèle SVM, nous avons observé que l'accuracy du modèle était trop basse et que nous n'atteindrions jamais de bon résultats avec ces modèles. Nous avons donc décidé de coder un MLP en utilisant la bibliothèque Pytorch pour essayer d'obtenir de meilleurs résultats. Pour ce modèle nous avons décidé de continuer avec l'encodage BOW. Ici on utilise un MLP avec 2 couches cachées et on se retrouve le modèle suivant :

$$\text{Input (i)} \rightarrow \text{1 Layer (i/10)} \rightarrow \text{2 Layer (i/100)} \rightarrow \text{Output (3)}$$

Avec "i" le nombre de mots gardé dans le vocabulaire. On entraine alors le modèle sur 15 epochs avec un optimiseur d'Adam, la fonction de perte "Cross Entropy", la fonction d'activation Sigmoïde, un learning rate de 0.001 et une taille de batch de 5000.

3.4 LSTM et GRU

Suite à l'utilisation des modèles recomandés par les consignes, et au vu des faibles résultats obtenus, nous voulions implémenter des réseaux de neurones plus complexes et dédiés à la NLP tels que les modèles LSTM et GRU, que nous avons entrainé sur 25 epochs avec un optimiseur d'Adam, la fonction de perte "Cross Entropy", la fonction d'activation Sigmoïde, un learning rate de 0.001 et une taille de batch de 512.

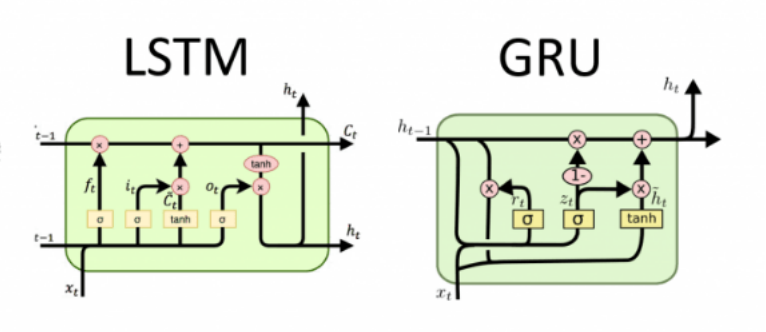

FIGURE 5 – Architectures des modèles LSTM et GRU

4 Méthodologie

Pour que l'entrainement se passe au mieux et pour tester l'efficacité des hyper paramètres, il nous à fallu diviser le jeu de données initial en un jeu d'entrainement et de validation. Ici, nous avons choisi d'allouer 20% des données à la validation et 80% à l'entrainement. De ce fait, on se retrouve avec un jeu d'entrainement de taille 824389 et un jeu de validation de taille 206098. Pour ce qui est du choix des hyper paramètres, nous avons décidé de prendre un nombre d'epochs suffisant, pour observer la convergence de l'accuracy et de la loss. Nous nous sommes également concentrés sur le nombre de mots du vocabulaire à garder et avons entrainé plusieurs modèles pour différentes tailles de vocabulaire. Également, nous avons réalisé plusieurs combinaisons entre les méthodes de nettoyage de données citées auparavant (ex : Normalisation et Lemmatisation, Normalisation seulement, etc). Enfin, la taille des batchs n'a pas réellement d'effets sur la précision du modèle mais plutôt sur sa complexité (Nous avons essayé plusieurs tailles possibles comme 512, 5000 ou 10000). Nous avons donc choisi des tailles relativement élevées suivant les modèles pour réduire le temps d'entrainement.

5 Résultats

5.1 Classifieur de Bayes Naïf

Nos premières recherches se sont portées sur le classifieur de Bayes Naïf et sur la recherche du nombre optimal de caratéristiques (mots du vocabulaire) ainsi que de la méthode adéquate de nettoyage des données. On voit ici que le meilleur résultat est obtenu pour 20000 features avec seulement de la normalisation. On voit néanmoins que pour un plus petit nombre de caractéristiques, les résutats sont à peu près les mêmes et le temps de calcul est plus faible. Pour ce qui est des méthodes de nettoyage des données on remarque que lorsque l'on ajoute les méthodes de mots vide, de lemmatisation ou de stemming, les résultats baissent. Plus de courbes et de tests sont présents en annexe pour attester de la véracité de ces propos.

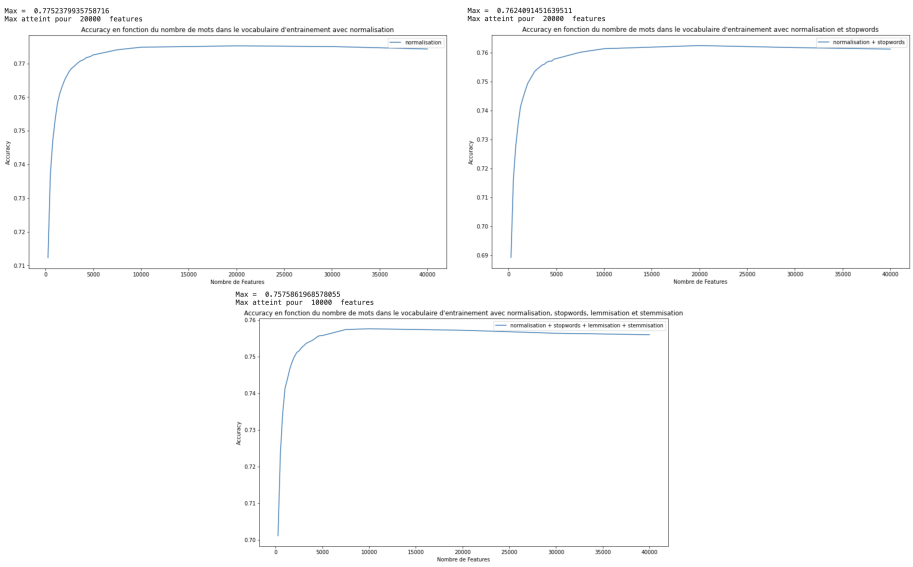

FIGURE 6 – Courbe d'accuracy en fonction du nombre de caractéristiques et pour des méthodes de nettoyage de données différentes

5.2 SVM

Avant de commencer à analyser les courbes, il est important de faire remarquer que notre SVM est entrainé sur seulement 1% des données et même avec ceci, le temps d'entrainement et la compléxité du modèle avec le kernel pour les chaînes de caractère sont trop élevés. Les résultats sur 1% des données sont comme nous l'attendions décevant. On remarque tout de même le même phénomène que pour le classifieur de Bayes avec l'ajout des méthodes de nettoyage des données. De plus contrairement à Bayes Naïf, le modèle SVM semble être plus efficace sur des petits nombre de caractéristiques (250).

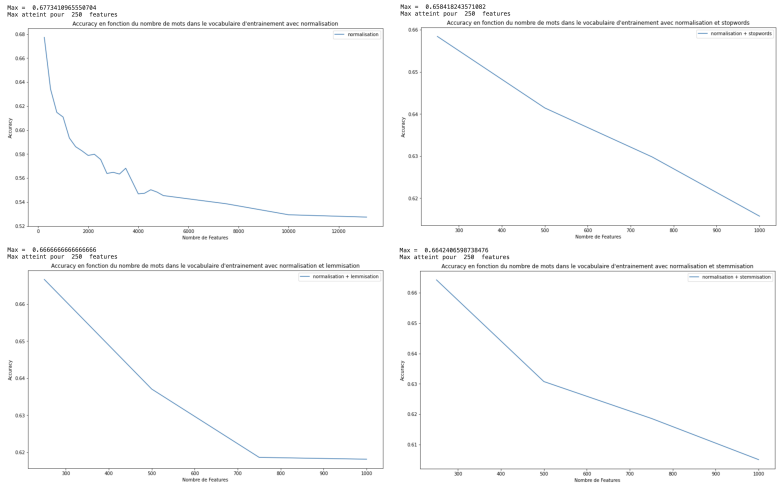

FIGURE 7 – Courbe d'accuracy en fonction du nombre de caractéristiques et pour des méthodes de nettoyage de données différentes

5.3 MLP

Au vu des résultats précédents, nous avons décidé de coder un MLP en utilisant uniquement la méthode de normalisation. Nous avons donc étudié l'impact de la variation du nombre de mots dans le vocabulaire sur l'accuracy et la loss. Pour ce qui est de l'entrainement, on obtient les meilleurs résultats avec le plus de caratéristiques. Au contraire, pour la validation on voit que les résultats sont similaires à partir de 4000 mots.

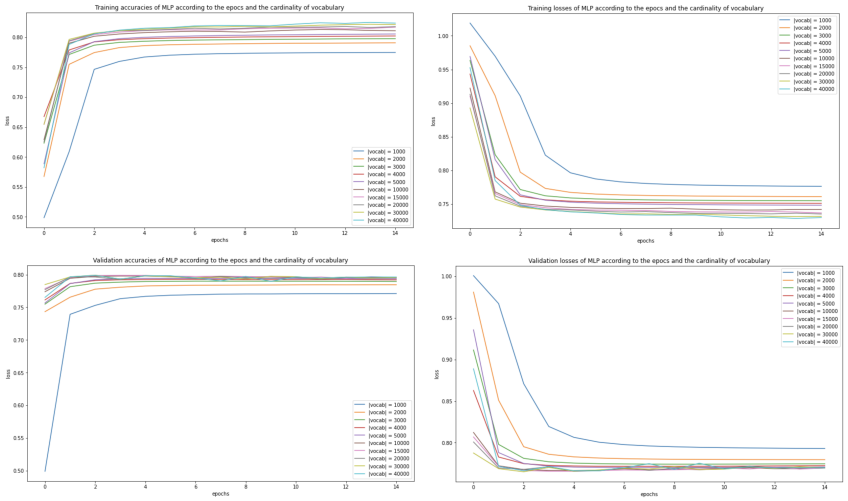

FIGURE 8 – Courbe d'accuracy et de loss en fonction du nombre d'epochs et de caractéristiques

5.4 LSTM et GRU

Pour les modèles LSTM et GRU, nous avons essayé de faire varier un grand nombre d'hyper paramètres pour obtenir des modèles différents. A l'instar des modèles ensemblistes vu en cours, notre objectif était de faire voter différents modèles pour une même prédiction et choisir la classe majoritaire obtenue. Nous avons donc fait varier la dimension de plongement, la dimension des couches cachées ainsi que la structure de la couche linéaire de sortie. Nous avons mis ci-dessous les meilleurs courbes des deux modèles différents. On voit que le LSTM croit légèrement et stagne après avoir "explosé", tandis que le GRU commence déjà avec une bonne accuracy (Validation > Entrainement) et stagne rapidement.

Meilleurs résultats Kaggle obtenus pour chaque algorithme :

	Bayes naif	SVM	MLP	LSTM	GRU	Modèle ensembliste
Résultats Kaggle	77.17%	64.66%	79% (val)	82.74%	82.80 (val.)	83.19%

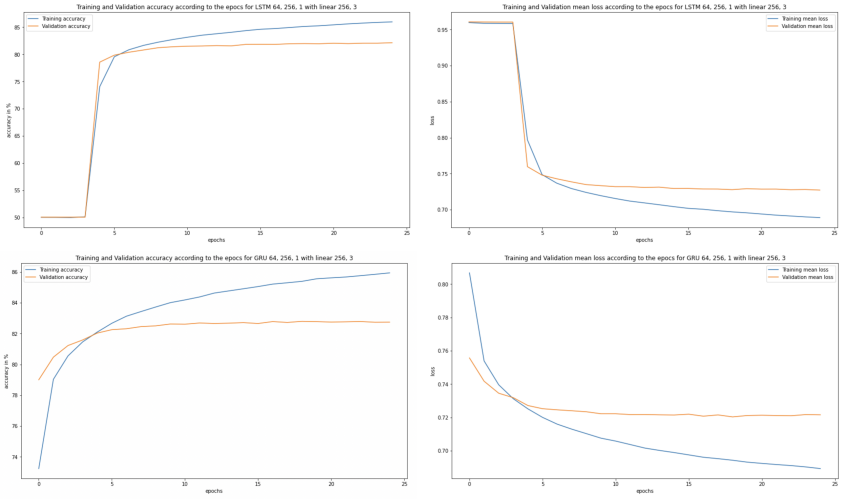

FIGURE 9 – Courbe d'accuracy et de loss en fonction du nombre d'epochs Pour les méthodes LSTM et GRU

6 Discussions

Au vu des résultats sur le jeu de données de test publique, on peut conclure que notre meilleure modèle est performant avec une accuracy de plus de 83%.

Cependant, en observant de plus près nos prédictions, nous constatons que la classe « neutral » n'a jamais été prédite.
Une cause assez logique serait le fait de la très faible fréquence de cette classe dans notre jeu de données d'entraînement : 0.008%. Nous avons pensé à deux possibilités d'amélioration :

6.1 Data Augmentation

La première technique pour améliorer les performances du modèle serait de faire de la Data Augmentation sur la classe sous représentée de notre jeu de données d'apprentissage. En effet, en NLP, il existe différentes techniques de Data Augmentation : nous avons pensé à la Back Translation et le Synonym Replacement.

6.1.1 Back Translation

Dans la Back Translation, nous traduisons les données textuelles dans une langue, puis les traduisons dans la langue d'origine. Cela peut aider à générer des données textuelles avec des mots différents tout en préservant le contexte des données textuelles.

Des API de traduction de langue comme Google Translate, Bing, Yandex sont utilisées pour effectuer la traduction.
Voici un exemple :

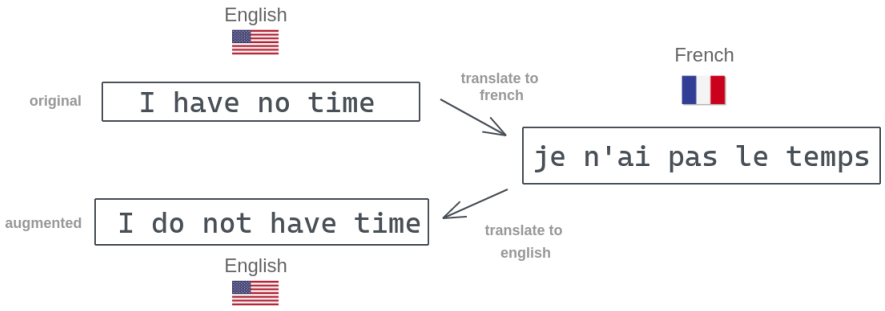

FIGURE 10 – Exemple de Back Translation

On peut ici voir que les phrases ne sont pas les mêmes mais leur contexte reste le même après la traduction.

6.1.2 Synonym Replacement

Dans le Synonym Replacement, on choisit au hasard n mots de la phrase qui ne sont pas des mots vides.
On remplace ensuite chacun de ces mots par un de ses synonymes choisis au hasard.

Exemple : Considérons la phrase "This article will focus on summarizing data augmentation techniques in NLP." en entrée.
On choisit alors par exemple 2 mots : "article" et "techniques", et on les remplace respectivement par des synonymes.
La phrase alors obtenue est la suivante :
"This write-up will focus on summarizing data augmentation methods in NLP."

6.1.3 D'autres techniques...

De plus nous avons pu voir que d'autres techniques existent telles que :

— Random Insertion
Cette méthode trouve un synonyme aléatoire d'un mot aléatoire dans la phrase qui n'est pas un mot vide. Ce synonyme est inséré dans une position aléatoire dans la phrase. Le procédé est répété n fois.

— Random Swap
Cette méthode choisit au hasard deux mots dans la phrase et échange leurs positions. Le procédé est répété n fois.

— Random Deletion
Cette méthode supprime au hasard chaque mot de la phrase avec une probabilité p.

6.2 Transformers

De plus, on pourrait exploiter le mécanisme d'attention du Transformer pour observer si il y a une amélioration des résultats ou non.

Le transformer est une nouvelle architecture qui vise à résoudre des tâches "Seq2Seq" tout en gérant facilement les dépendances à longue distance.
Le Transformer permet de calculer les représentations d'entrée et de sortie sans utiliser de RNN ou de convolutions alignées sur la séquence et cela repose entièrement sur la self-attention.

7 Division des contributions

Voici les contributions de chaque membre de l'équipe pour les composantes les plus importantes de la compétition :

— Définir le problème : Thomas et Gabin
— Développer la méthodologie : Thomas et Nazim
— Coder la solution : Nazim et Gabin
— Améliorer les performances et les résultats : Nazim et Gabin
— Effectuer l'analyse des données : Thomas et Gabin
— Rédiger le rapport : Nazim et Thomas

Nous déclarons par la présente que tous les travaux présentés dans ce rapport sont ceux des auteurs.

Références

[1] Image de l'encodage BOW :
`https://www.askpython.com/python/examples/bag-of-words-model-from-scratch`

[2] Image de l'encodage par plongement :
`https://lena-voita.github.io/nlp_course/word_embeddings.html`

[3] Idée d'implémenter un MLP, un LSTM et un GRU :
Le cours IFT 6135 : Representation Learning, présenté par Aishwarya Agrawal
`https://sites.google.com/mila.quebec/ift6135-a2022/plan-de-cours`

[4] Documentation Pytorch
`https://pytorch.org/docs/stable/index.html`

[5] Documentation Scikit Learn
`https://scikit-learn.org/stable/user_guide.html`

8 Annexe

8.1 Annexe graphes modèle de Bayes

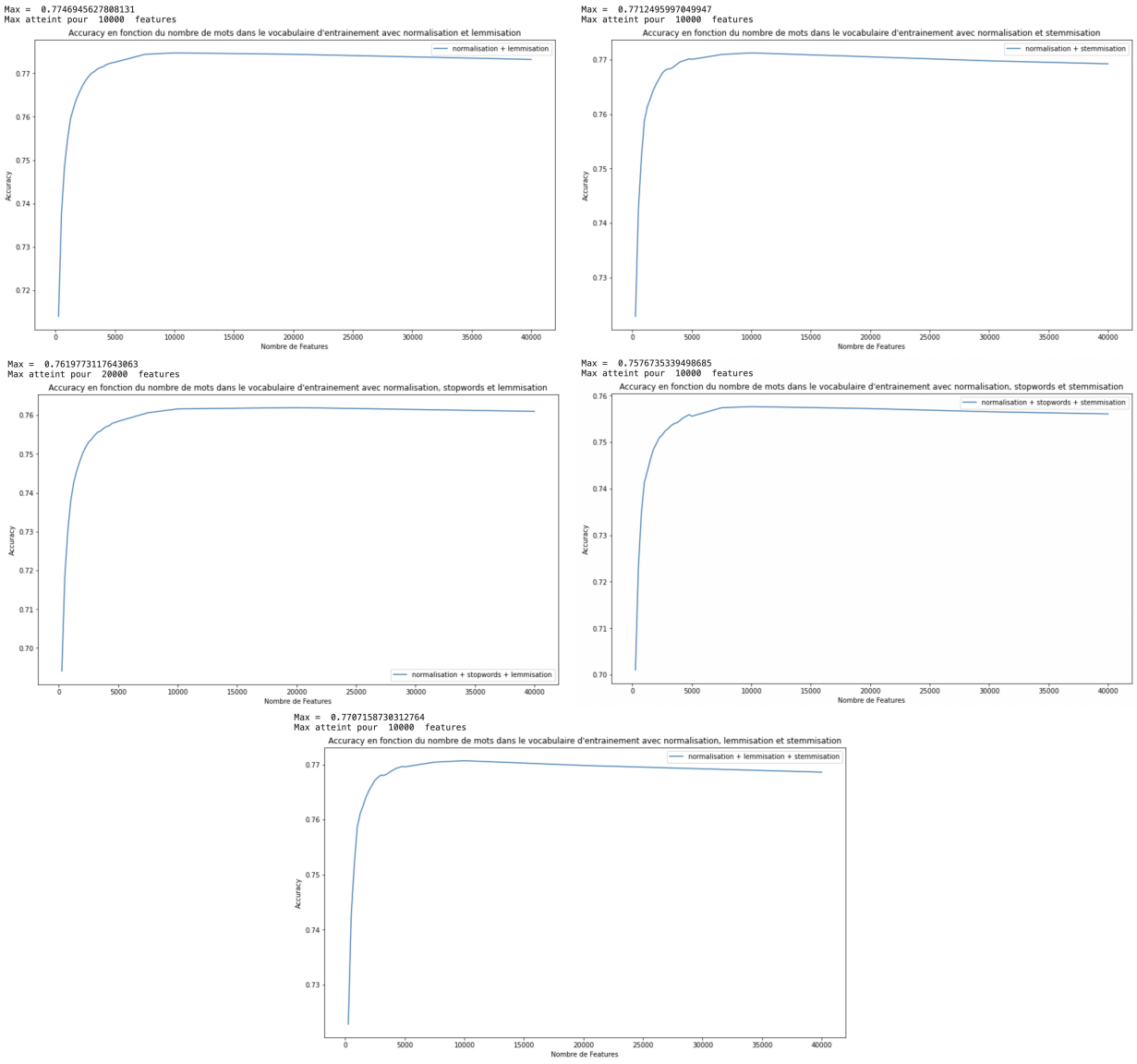

FIGURE 11 – Courbe d'accuracy en fonction du nombre de caractéristiques et pour des méthodes de nettoyage de données différentes

8.2 Annexe graphes modèle LSTM et GRU

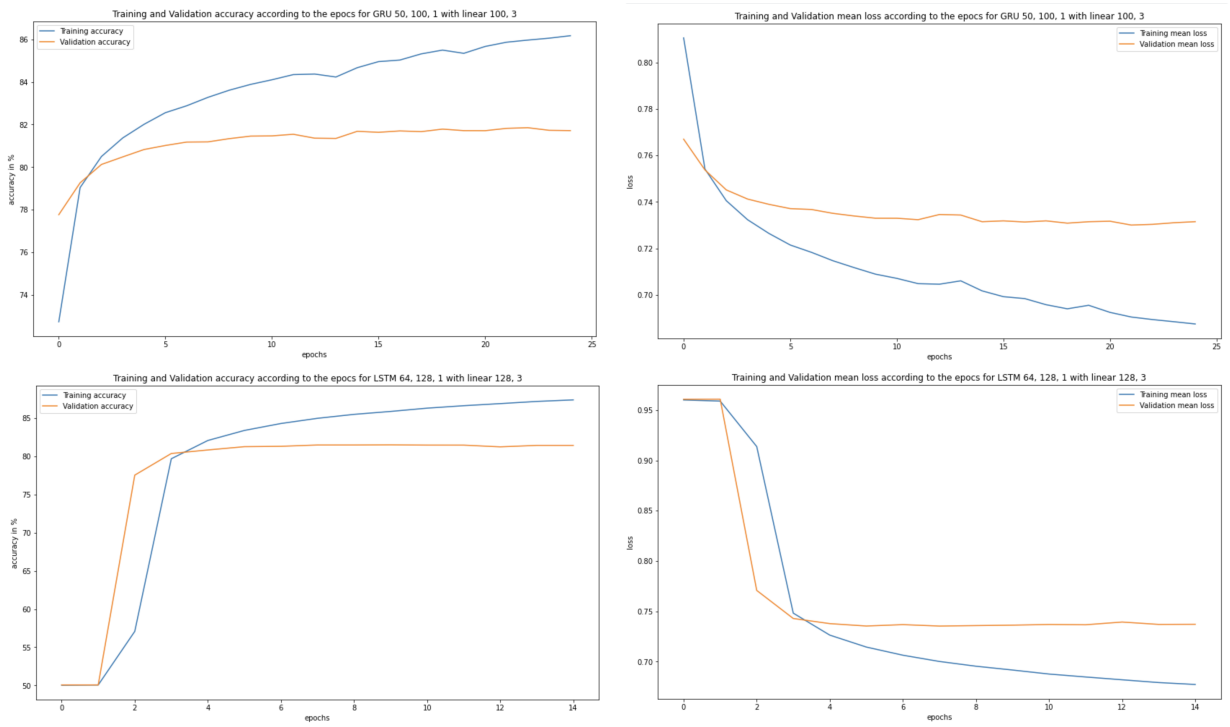

FIGURE 12 – Courbe d'accuracy et de loss en fonction du nombre d'epochs et de caractéristiques