Projet 8 : Participez à une compétition Kaggle

Maby Antoine
Janvier 2022

Sommaire

Chapitre 1

Introduction

Kaggle est une plateforme qui organise des compétitions en data science et qui récompense les meilleurs analystes internationaux. L'objectif de ce projet est de participer à une de ces compétitions en passant par toutes les étapes de l'analyse : récupération, nettoyage des données, analyse exploratoire, création de plusieurs modèles et mesure de leurs performances, etc.

Kaggle est conçue comme une plateforme collaborative. Ce projet est l'occasion de participer à cette communauté. Pour cela, nous pourrons nous appuyer sur les kernels partagés par d'autres participants. En retour, nous partagerons également notre travail, sous forme d'un kernel Kaggle présentant des éléments pertinents : un point d'analyse exploratoire qui vous a servi dans la phase de modélisation, ou l'un de vos modèles. La communauté pourra ainsi commenter et en discuter.

La compétition que nous choisissons pour ce projet est Jigsaw Rate Severity of Toxic Comments : Rank relative ratings of toxicity between comments. C'est une compétition portant sur la toxicité de commentaires sur Wikipédia. Nous l'avons choisi car elle nous permettra de mettre en place certaine de nos compétences aquises au cours du projet 5, mais aussi de les développer, puisque le projet 5 était le seul portant sur des données textuelles. Nous détaillerons les contours de la compétitions dans une prochaine partie.

Chapitre 2

Description de la compétition

Le sujet de la compétition est l'examination de commentaires. Si nous demandons à une personne d'examiner des commentaires, sans contexte, lequels sont les plus toxiques ou lequels sont les plus innoffensifs, la tâche est difficile. De plus, chaque individu peut avoir sa propre barre de toxicité. Une idée pourrait être de faire un vote à la majorité pour décider. Seulement cette solution implique une perte d'information. Les auteurs du concours ont demandé à des individus de choisir entre deux commentaires lequel est le plus toxique. Cette idée semble résoudre une des problèmatiques du sujet. Cependant, lorsque les deux commentaires ne le sont pas, le choix semble se porter vers le hasard, mais lorsqu'un commentaire est significativement plus toxique, les résultats entre les individus devraient concorder.

Dans ce concours, il est demandé de trier un ensemble d'environ quatorze mille commentaires. Des paires de commentaires ont été présentées à des évaluateurs experts, qui ont marqué le commentaire le plus nocif, chacun selon sa propre notion de toxicité. Dans ce concours, lorsque vous fournissez des scores pour les commentaires, ils seront comparés à plusieurs centaines de milliers de classements. L'accord moyen avec les évaluateurs déterminera le score individuel. De cette façon, l'espoir est de se concentrer sur le taux de toxicité des commentaires du plus inoffensif au plus scandaleux, où le milieu compte autant que les extrêmes.

Dans ce concours, nous disposions des datasets suivants : un dataset de Validation composé de commentaires comparés deux par deux et répartis entre le moins toxique et le plus toxique pour tester nos algorithmes ainsi que d'un dataset présentant la forme des soumissions. Et enfin, Comments, les commentaires que nous annonterons à la fin du projet. Ainsi, il n'est pas fourni de dataset prédéfinit pour entraîner nos modèles. Ce n'est pas le premier concours portant sur la toxicité des commentaires sur Wikipédia, nous irons nous servir dans ceux-ci pour entraîner nos modèles. Le choix s'est porté sur deux datasets. Le dataset train de Toxic Comment Classification Challenge qui servira de base pour l'entraînement de tous les modèles. La base de données Augmentation de Jigsaw Unintended Bias in Toxicity Classification sera utile pour le pré-entraînement d'un modèle de Sentence Encoder.

	Validation(paire de commentaires)	Comments	Train	Augmentation
Nombre de commentaires	30108	7537	159571	1999516

Chapitre 3

Analyse Exploratoire et Nettoyage du Corpus

Passons à la partie analyse exploratoire puis des différents nettoyages effectués au corpus.

3.1 Analyse Exploratoire

Dans un premier temps, décrivons le dataset de Train. Il est composé des commentaires et de 6 classes caractérisant les commentaires. Ces 6 classes sont : toxic, severe toxic, obscene, threat, insult et indentity hate. Pour analyser la présence de ces classes, nous sommons toutes les classes pour chaque commentaire. Sur la Figure (3.1), on peut voir que la plupart des commentaires n'ont pas de classes. Sur les 150 000 commentaires composant ce dataset, seulement 16 000 ont au moins 1 classes non nulles.

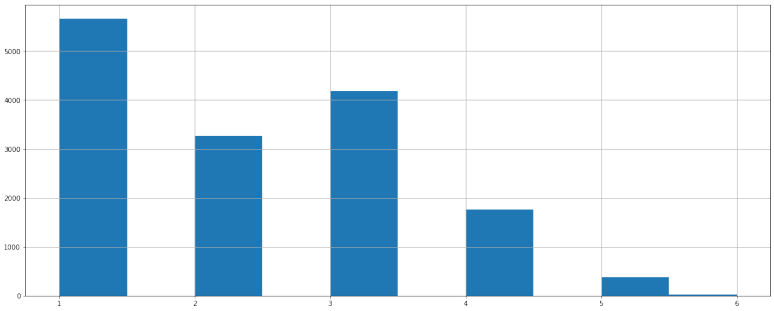

FIGURE 3.1 – Sommes de toutes les classes pour chaque commentaire

À partir de des commentaires, il est possible de créer quelques nouvelles features qui sont : le nombre de mots, le nombre de mots uniques, le nombre de stopword, le nombre de ponctuations, la longueur moyenne des mots et le nombre de caractères. Analysons quelques-unes des ces nouvelles variables.

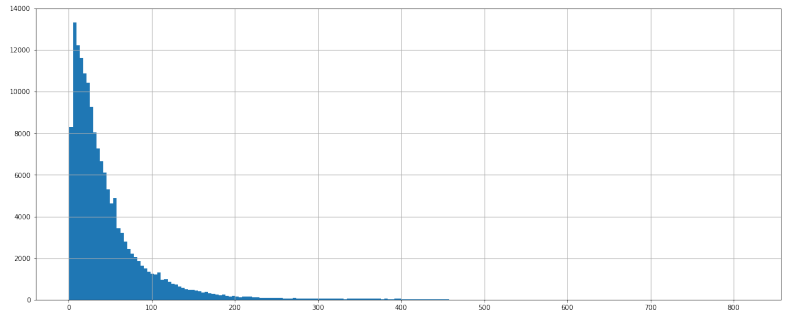

FIGURE 3.2 – Distribution du nombre de mots unique

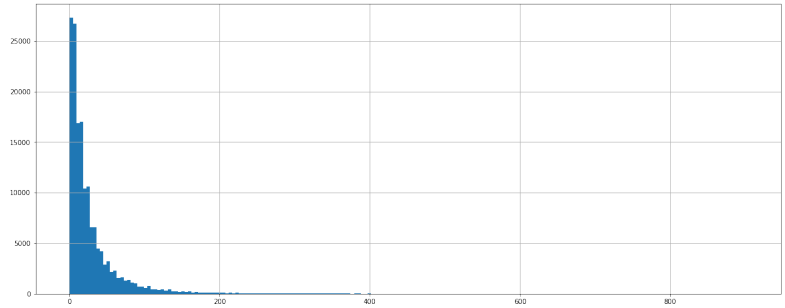

FIGURE 3.3 – Distribution du nombre de stopwords

Il est intéressant que le pic de la distribution du nombre de mots uniques soit situé pour plus de mots que le pic de la distribution de stopwords. Cela se reporte également sur la moyenne, ainsi nous devrions avoir toujours des mots significatifs pour une majorité de commentaires. La dernière partie à souligner concerne le nombre de ponctuation. Logiquement 50% des commentaires ont un nombre inférieur à 8 ponctuations. Mais certains commentaires atteignent plus de 5000 ponctuations. Cette valeur est intéressante car elle pourrait être un indicateur de la toxicité du commentaire.

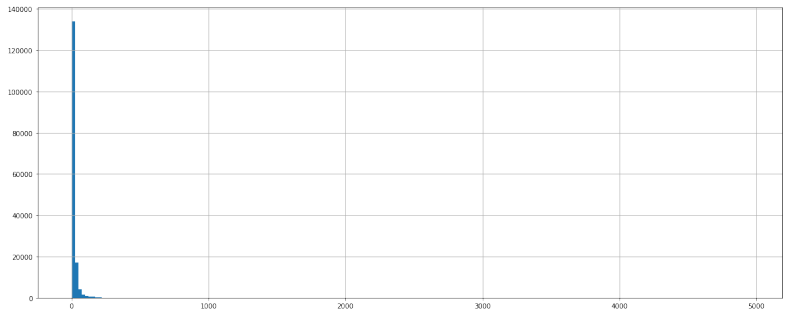

FIGURE 3.4 – Distribution de la ponctuation

Après la description de notre corpus, passons à la partie nettoyage.

3.2 Nettoyage du Corpus

Dans un premier temps, utilisons la même routine que lors du projet 5 :

— Suppression des bannières HTML
— Mettre tout le texte en minuscules
— Supprimer les caractères Unicode
— Suppression des espaces supplémentaires
— Suppression de la ponctuation
— Suppression des liens
— Supprimer les nombres

Dans un deuxième temps, nous appliquons le package Beautifoulsup pour completer le nettoyage. Ensuite, nous utilisons une routine créee par un participant du concours. Celle-ci corrige beaucoup de contractions courantes sur internet ou l'utilisation de chiffres à la place de lettres. [1]

La troisième étape dépendra du Sentence Encoder. Pour Tfidf, il est nécessaire de passer par les processus de Lemmatisation, suppressions des stopswords et Tockenisation, sinon les Sentence Encoder n'auront besoin que des deux premières étapes.

Ces différentes étapes seront répétées pour les dataset tel que Train, Validation, Augmentation et Comments. Passons maintenant à la recherche d'une Baseline.

Chapitre 4

Baseline

Dans cette partie, nous rechercherons la Baseline, la méthode appliquée dans la suite et que nous essayerons d'améliorer. Pour tous les modèles, un Tfidf sera notre Sentence Encoder avec les processus de nettoyages décrits précédement. Les performances de chacun seront testées sur le Dataset de Validation en comparant les scores des commentaires les plus ou moins toxiques.

4.1 Random Forrest Classifier

La méthode est un Random Forrest sur les classes du Dataset de Train. Nous appliquons un Random Forrest multi-task. Les classes sont Toxic, Severe Toxic, Obscene, Threat, Insult, Identity hate. Une fois les prédictions effectuées, une addition de chaque classe des commentaires définira le score entre 0 et 6. C'est ce score qui déterminera si celui de Less Toxic est bien plus faible que le score de More Toxic. Nous avons effectué un rapide fine tuning de la Random Forrest.

Les performances obtenues sont la moyenne des bonnes prédictions, un score plus grand pour la colonne more toxic comparé à la colonne less toxic du Dataset Validation.

	Random Forrest	Lightgbm Regression	Lightgbm Classification
Score	0.46	0	0

4.2 Régression Lightgbm

Cette fois-ci, nous utilisons la régression Lightgbm ([2]) qui sera effectuée directement sur la somme des classes pour chaque commentaire. Ainsi, les commentaires seront classés de 0 à 6 et la prédiction du modèle donnera les scores pour less toxic et more toxic.

Les performances obtenues sont la moyenne de bonnes prédictions, c'est-à-dire un score plus grand pour la colonne more toxic comparé au score de la colonne less toxic du Dataset Validation.

	Random Forrest	Lightgbm Regression	Lightgbm Classification
Score	0.46	0.65	0

4.3 Classification Lightgbm

Dans cette partie, nous appliquons une méthode de classification binaire Toxic ou non Toxic sur les données. L'idée est venue car le Dataset Augmentation est d'une grande taille. Mais, dans celui-ci, la toxicité est une note compris entre 0 et 1. Ainsi, pour tout score de toxicité non nulle, la classe sera à 1 sinon 0. Les dataset de Train et d'Augmentation ont été pris pour cette méthode. Le score sera donné par la probabilité que le commentaire soit Toxic ou non.

Les performances obtenues sont la moyenne de bonnes prédictions, c'est-à-dire un score plus grand pour la colonne more toxic comparé au score de la colonne less toxic du Dataset Validation.

	Random Forrest	Lightgbm Regression	Lightgbm Classification
Score	0.46	0.65	0.57

Il semblerait que la meilleure méthode soit la régression Lightgbm sur la somme des classes. Nous poursuivrons avec cette méthode en tentant de l'améliorer.

Chapitre 5

Sentence Encoder

Dans cette partie, nous essayerons différents Sentence Encoder. Le but de ces routines est de créer -à partir des commentaires- des vecteurs de features. Plusieurs de ces modèles seront testés et combinés pour analyser les performances. La méthode appliquée est la Lightgbm sur la somme des classes. Les performances sont prises sur la moyenne des commentaires More Toxic avec un plus grands score que les commentaires Less Toxic correspondant.

5.1 Universal Sentence Encoder

Dans cette partie, nous testons Universal Sentence Encoder ([3]), nommé USE dans la suite. Les performances sont :

	USE	All Mini L12	All Mini L12 pré-train	Tfidf et All Mini L12	USE et All Mini L12
Score	0.6858	0	0	0	0

Il est déjà notable que les performances d'Universal Sentence Encoder soient meilleures que celles de la Tf-idf.

5.2 Sentence Transformer

Dans cette partie, nous utilisons Sentence Transformer([4]). Plusieurs sentences encoder étaient possible. Le choix s'est dirigé vers un compromis entre de la rapidité de calculs et les meilleures performances possible. Ainsi, nous choisissons All Mini L12. Les performances de Lightgbm combiné à All Mini L12 sont :

	USE	All Mini L12	All Mini L12 pré-train	Tfidf et All Mini L12	USE et All Mini L12
Score	0.6858	0.6857	0	0	0

Les performances sont très proches de celles obtenues avec Universal Sentence Encoder.

5.3 Sentence Transformer pré-train

Sentence Transformer permet également d'adapter ces modèles en effectuant un fine tuning sur nos propres données. L'idée du fine tuning est de comparer des paires de commentaires. Dans notre cas, nous l'entraînons en donnant des paires nulles, c'est-à-dire un commentaire non toxic/toxic et des paires positives (deux commentaires Toxic). Pour l'entraîner, nous nous sommes servis du Dataset Augmentation, ainsi que du modèle ALl Mini L12. Les performances de Lightgbm combiné à All Mini L12 pré-train sont :

	USE	All Mini L12	All Mini L12 pré-train	Tfidf et All Mini L12	USE et All Mini L12
Score	0.6858	0.6857	0.6824	0	0

Les performances sont inférieures à celles de Sentence Transformer sans pré-train. Ceci est peut-être dû à un mauvais pré-entraînement. Soit le nombre de paires n'étaient pas suffisant, soit notre définition des paires n'était pas la bonne.

5.4 Tfidf et Sentence Transformer

Nous essayons dans cette partie d'assembler plusieurs méthodes pour tester si les performances s'améliorent. Nous concatenons le TF-idf et le Sentence Transformer. Les performances de Lightgbm combiné Tf-idf et à All Mini L12 sont :

	USE	All Mini L12	All Mini L12 pré-train	Tfidf et All Mini L12	USE et All Mini L12
Score	0.6858	0.6857	0.6824	0.6815	0

Les performances sont inférieures à celles de Sentence Transformer sans pré-train.

5.5 Sentence Transformer et Universal Sentence Transformer

Dans cette section, nous testons l'assemblage de Sentence Transformer et Universal Sentence Transformer. Les performances de Lightgbm combinées à Universal Sentence Encoder ainsi qu' à All Mini L12 sont :

	USE	All Mini L12	All Mini L12 pré-train	Tfidf et All Mini L12	USE et All Mini L12
Score	0.6858	0.6857	0.6824	0.6815	0.6868

Les performances sont supérieures à celles de Sentence Transformer sans pré-train. Nous utiliserons la combinaison de Universal Sentence Encoder avec Sentence Transformer dans la suite. Passons maintenant à de nouvelles améliorations de notre modèle

Chapitre 6

Amélioration du modèle choisi

Dans cette partie, nous voulons améliorer le modèle de Lightgbm avec Sentence Universal Encoder et Sentence Transformer. D'abord, nous nous focaliseons sur les hyper-paramètres de la LightGBM, puis nous intégrerons les features que nous avons créees pour finalement, tenter d'introduire des poids dans les classes.

6.1 Finetuning de Lightgbm

Nous réalisons un finetuning de LightGBM. Les paramètres modifiés principalement sont num leaves, la profondeur des nœuds et le nombre minimum de data dans un nœud. Une fois le fine tuning effectué par tatonnement, nous obtenons les performances suivantes :

	Base	Finetune	Feature	Poids 1	Poids 2
Score	0.6858	0.6904	0	0	0

Nous améliorons les performances de 0.005 grâce au finetuning. Passons à de nouvelles pistes d'amélioration.

6.2 Ajout des features créees

Au début de notre projet, nous avions crée les features suivantes : le nombre de mots, le nombre de mots uniques, le nombre de stopword, le nombre de ponctuations, la longueur moyenne des mots et le nombre de caractères. Intégrons les au modèle pour voir si elles augmentent les performances.

	Base	Finetune	Feature	Poids 1	Poids 2
Score	0.6858	0.6904	0.6872	0	0

Les performances sur les données de validations sont plus faibles. Nous choisissons de ne pas les ajouter.

6.3 Poids des classes

Dans cette partie, nous ajoutons des poids aux différentes classes. Jusqu'à maintenant, le poids de chaque classes est de 1. Nous testons plusieurs répartitions.

Dans un premier temps, analysons les poids obtenus sur un Kernel de la compétition. [1]

	Toxic	Severe Toxic	Obscene	Threat	Insult	Identity Hate
Poids	0.32	1.5	0.16	1.5	0.64	1.5

	Base	Finetune	Feature	Poids 1	Poids 2
Score	0.6858	0.6904	0.6872	0.6821	0

Les performances sont finalement inférieures à celles obtenues jusqu'à maintenant. Testons à present l'ajustement de nos propres poids par tatonnement.

	Toxic	Severe Toxic	Obscene	Threat	Insult	Identity Hate
Poids	0.5	2	0.5	2	0.5	2

	Base	Finetune	Feature	Poids 1	Poids 2
Score	0.6858	0.6904	0.6872	0.6821	0.6833

Une nouvelle fois les performances sont en deça des résultats obtenus jusqu'à maintenant. L'importance des poids sur les données de validation ne semblent pas être une bonne voie.

Chapitre 7

Conclusion

Cette compétition a pour but de scorer la toxicité des commentaires. Afin de ne pas être dépendant de la barre de toxicité de chacun, l'objectif était d'avoir un score à comparer entre deux commentaires, d'un supposé plus toxique que l'autre. Après une rapide étude exploratoire et un nettoyage des données, nous avons recherché et mis en place plusieurs méthodes dans le but d'avoir un score de toxicité associé des commentaires. Finalement, le choix s'est porté vers Lightgbm sur une somme de classes.

Une fois cette Baseline trouvée, nous cherchions à améliorer les performances. Pour cela, l'idée était de trouver le meilleur Sentence Encoder possible. Après cette étude, le choix s'est porté sur la combinaison entre Universal Sentence Encoder et Sentence Transformer. Finalement, nous avons tenté plusieurs petites améliorations qui n'ont pas porté leurs fruits sur les données de Validation. Malgré tout, certaines de ces idées seront testées sur la compétition. Il semble que nous approchions des performances maximales obtenus sur le Dataset de Validation. Nous ne pouvons tester que sur les kernels de la compétitions sur les donnnées de Tests.

Ce projet sur la toxicité des commentaires nous a permis d'augmenter nos connaissances sur le traîtement des données textuelles mais aussi d'apprendre à commencer un projet comme mettre en place notre propre méthode de recherche ou encore choisir des Datasets, une méthode, et tenter de l'améliorer. Nous avons décrit nos recherches dans ce document, et il est possible de retrouver un Notebook sur un Git mis en place pour le Projet. [5]

Bibliographie

[1] VitaLeeY. https://www.kaggle.com/vitaleey/tfidf-ridge#Please-upvote-if-you-like-my-work-:), 2022.

[2] Microsoft Corporation. https://lightgbm.readthedocs.io/en/latest/index.html, 2022.

[3] Daniel Cer, Yinfei Yang, Sheng yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. Universal sentence encoder, 2018.

[4] Nils Reimers and Iryna Gurevych. Sentence-bert : Sentence embeddings using siamese bert-networks, 2019.

[5] Antoine Maby. https://github.com/Ulytonio/Projet-8-Participe-a-une-competition-Kaggle, 2022.