Analyse de sentiment d'un tweet : 3 approches

J'ai récemment été amené à réaliser un déploiement sur le cloud d'un modèle de NLP pour prédire le sentiment de tweets (positif ou négatif). Je vais donc présenter les trois modèles de complexités différentes que j'ai implémenté et comparé.

Le but est d'aider une entreprise à anticiper les "bad buzz", la priorité est donc de détecter les sentiments négatifs.
J'ai choisi le score F-2 comme métrique pour départager les modèles car elle priorise le rappel, ce qui est adapté à notre problème, mais n'ignore pas non plus la précision.

Prenons d'abord le plus simple des trois modèles : la régression logistique.
J'utiliserai son score F-2 comme point de référence pour les modèles plus complexes, ce qui permet de mieux évaluer leurs performances.

Avant de présenter les tweets au modèle, je les pré-traite de la manière suivante :
Je passe d'abord l'ensemble du texte en minuscule et convertis l'argot en langage classique grâce au module "contractions". Ces étapes ne sont pas nécessaires, mais permettent de simplifier le corpus.
Je tokenise ensuite les tweets avec le module "nltk" pour obtenir une liste de tokens.
Sur ces tokens j'applique une lemmatisation, ce qui simplifie encore les tweets.
Finalement, j'utilise la méthode bag of words pour transformer chaque liste de tokens en vecteur lisible par le modèle.

Les données sont prêtes, je peux désormais les séparer en trois groupes :
jeux d'entraînement (80%), de validation (10%) et de test (10%).

Le modèle de régression logistique ayant peu de paramètres, l'entraînement se fait en moins de 10 minutes sur CPU.
J'obtiens un score F-2 = 0.77 sur le jeu de test, ce que je trouve impressionnant pour un modèle si simple face à une tâche que je pensais complexe.

Comparons maintenant ce résultat à celui du modèle suivant : un réseau de neurones récurrent.
Il est composé d'une couche de plongement de mot en entrée, de trois couches successives de LSTM, et d'une couche de neurones classiques en sortie.

Contrairement à la méthode bag of words qui prend seulement en compte combien de fois un mot est présent dans le texte, le plongement de mot permet de convertir un mot en un vecteur de coordonnées dans un espace donné (300 dimensions pour fastText par exemple). Tous le corpus se retrouve inscrit dans cet espace, on peut donc étudier la distance entre les mots pour comprendre leurs relations. Le modèle bénéficie donc de "contexte" pour chaque mot compte tenu de sa position dans l'espace.

En parlant de contexte : le réseau de neurones récurrent est séquentiel, c'est-à-dire qu'il prend en compte l'ordre des mots (ou plutôt les coordonnées des mots) de la phrase qu'il reçoit.
Tout ceci couplé à un plus grand nombre de paramètres à entraîner, on peut s'attendre à un meilleur

résultat que la régression logistique.

Ce modèle comporte environ 60 000 paramètres à entraîner et nécessite plusieurs époques pour converger, ce qui augmente drastiquement le temps d'entraînement.
En utilisant cette fois-ci un GPU, le temps d'entraînement est de 2 minutes par époque.
Une dizaine d'époques suffisent à atteindre une convergence de la fonction de perte.

J'ai testé deux plongements de mots pré-entraînés, GloVe et fastText.
La différence de score est très faible : - score F-2 avec Glove = 0,817
- score F-2 avec fasText = 0,829

On constate bien une amélioration du score, même si je m'attendais à un plus grand écart du résultat de la régression logistique.

Passons maintenant au plus complexe : un modèle de type BERT.
Ce modèle est basé sur l'architecture "transformer" et par rapport au modèle précédent, ajoute notamment le mécanisme d'attention. À sa publication en 2018, BERT est devenu le modèle le plus performant sur les tâches de prédiction de mot et d'analyse de sentiments (mais pas sur la génération de texte, ce prix revient pour l'instant aux modèles de type GPT).

J'utilise le modèle DistilBERT qui vise à conserver les performances du modèle initial tout en réduisant sa taille et par conséquent son temps d'entraînement et d'inférence.
DistilBERT retient la grande majorité des performances (97% sur le benchmark GLUE) tout en réduisant le nombre de paramètres de 40%.
Pour plus de précisions à ce sujet, lire cette publication : DistilBERT.pdf (ysu1989.github.io)

Le modèle est pré-entraîné sur une très grande quantité de données, il suffit d'ajouter une couche de classification en sortie et de l'entraîner sur notre tâche en quelques époques avec un taux d'apprentissage faible (ici 0.00002).

Par curiosité, j'ai retiré toute les étapes de pré-traitement du corpus pour voir comment se débrouillerait le modèle avec toute la complexité présente naturellement.

On peut constater l'utilité de la réduction de taille, car du haut de ses 66 millions de paramètres, DistilBERT fini son entraînement de 3 époques en environ 14 heures sur GPU (soit presque 5 heures par époque). Cependant, le temps d'inférence est assez rapide : moins d'une seconde sur CPU.

Avec un score F-2 = 0.87, on voit encore une fois une amélioration qui le place (sans surprises) en tête.

Voici un tableau comparatif des scores et temps d'entraînement des différents modèles :

	F-2 score	Total train time (min)	1 epoch train time (min)	On GPU
Linear regression	0,770	4	4	NO
Neural network	0,829	40	2	YES
DistilBERT	0,865	825	275	YES

Les modèles de type BERT ont la réputation d'être très performants sur ce type de tâche.
J'émets l'hypothèse que se concentrer sur la qualité des données, des labels et du pré-traitement sera plus efficace pour continuer d'améliorer notre score que de modifier le modèle.