UFR 6

Université Montpellier III

Mémoire

Ivan Can Arisoy

Master 1 MIASHS

Natural Language Processing with Disaster Tweets

Predict which Tweets are about real disasters and which ones are not

Table des matières

Liste des figures **iii**

Remerciements **v**

Résumé **v**

1 Introduction **1**
1.1 Contexte 1
1.2 Données 2
1.3 Problématique 2
1.3.1 Hypothèses 3
1.4 Objectifs 3

2 Méthodologie **4**
2.1 Outils et technologies utilisés 4
2.1.1 BERT-base 6
2.1.2 BERTweet-base 6
2.2 Pré-traitement des données 7
2.2.1 Tokenization 7
2.3 Analyse descriptive 8
2.4 Transfer Learning et le Masked Language Modeling (MLM) 10
2.5 Fine-tuning 11
2.5.1 Hyperparamètres 11
2.5.2 Optimisation du modèle 12
2.5.3 Fonction de coût 13
2.6 Critères d'évaluation 13

3 Résultats et Discussions **14**
3.1 Résultats sans MLM 14
3.2 Résultats avec MLM 15
3.3 Réduction de la taille de lot 15
3.3.1 Extension des tests à d'autres modèles 16
3.4 Analyse t-SNE 17

3.5 Validation croisée 19

4 Conclusion et Perspectives **20**
4.1 Compétences acquises 20
4.2 Adéquation avec mes études 20
4.3 Difficultés rencontrées 21
4.4 Perspectives 21

Bibliographie et Ressources **21**

Annexe **23**

A Détails Supplémentaires **23**
A.1 Détails des résultats sans MLM 23
A.2 Détails des résultats avec MLM 26

Table des figures

2.1 Logo Hugging Face. 4
2.2 Logo PyTorch 4
2.3 Logo Jupyter, Google Colab. 5
2.4 Logo Notion. 5
2.5 BERT-base et BERT-large 6
2.6 Distribution des labels 8
2.7 Distribution de la longueur des tweets 9
2.8 top-20 mots les plus courants dans les tweets désastres 9
2.9 Algorithme de Adam et AdamW 12

3.1 t-SNE modèle avec MLM 18
3.2 t-SNE modèle sans MLM 18
3.3 t-SNE BERTTweet 19
3.4 Les folds de la validation croisée 19

A.1 Détails et visualisation des performances d'entraînement : **BERT-base batch size 32** 23
A.2 Détails et visualisation des performances d'entraînement : **BERTweet-base batch size 32** 24
A.3 Détails et visualisation des performances d'entraînement : **BERT-base batch size 8** 25
A.4 Détails et visualisation des performances d'entraînement : **BERTweet-base batch size 8** 25
A.5 Détails et visualisation des performances d'entraînement : **BERT-base avec MLM batch size 32** 26
A.6 Détails et visualisation des performances d'entraînement : **BERT-base avec MLM batch size 8** 26

Remerciement

Je tiens particulièrement à remercier Monsieur Maximilien Servajean pour l'approbation du projet et les conseils qui m'ont permis de le développer. Je voudrais adresser mes remerciements à l'équipe pédagogique de mon université, qui m'a formé et m'a donné des conseils en dehors de ce projet, ce qui m'a également aidé à progresser tout au long de l'année.

Résumé

Ce mémoire retrace mon expérience dans le domaine du traitement automatique du langage naturel à travers une compétition Kaggle. Mon projet s'articule autour de l'évaluation de l'efficacité de diverses techniques pour les modèles d'apprentissage automatique dans une tâche de classification, en particulier l'utilisation de la technique de pré-entraînement par masquage (Masked Language Modeling).

L'impact des différentes techniques est démontré sur les modèles comme BERT-base et BERTweet-base pour la classification des tweets. De plus, ce mémoire met en évidence l'importance de la sélection et de la validation des modèles pour améliorer leurs performances. Finalement, j'explique comment ce projet m'a permis d'appliquer la théorie en pratique et comment cette expérience m'aidera dans le futur.

Chapitre 1

Introduction

1.1 Contexte

Le traitement automatique du langage naturel (TALN) connaît une évolution rapide depuis l'introduction de l'apprentissage supervisé et semi-supervisé au début du XXIe siècle. Avec l'arrivée des *word embeddings*, il est devenu possible de capturer non seulement des aspects statistiques, mais aussi de capturer les mots dans un espace de grande dimension. Ce n'est qu'après l'émergence récente des modèles *transformers* et du mécanisme d'attention [1] qu'il est devenu possible de représenter la significativité et la similarité contextuelle des mots, ce qui se rapproche de la compréhension humaine.

Cette année, j'ai eu l'occasion d'enrichir mon expérience dans ce domaine grâce à ce projet universitaire. Personnellement, je trouve que c'est une opportunité unique de pouvoir étudier ce domaine scientifique jeune et prometteur, qui aura sans doute une importance considérable pour moi à l'avenir. L'une des plateformes permettant d'explorer et de se familiariser avec le TALN et la science des données en général est Kaggle. Kaggle propose des jeux de données variés créés par la plateforme et ses utilisateurs. Elle organise également des compétitions accessibles, qui offrent la possibilité de participer librement selon des règles et des critères d'évaluation prédéfinis. Ces compétitions encouragent une approche méthodologique flexible et, en proposant un système d'évaluation externe, permettent de se concentrer sur la gestion de projet et les stratégies nécessaires.

La compétition que j'ai choisie s'intitule : **"NLP with Disaster Tweets : Predict which Tweets are about real disasters and which ones are not."** [2] L'idée est de prédire, de manière binaire, si un tweet concerne un désastre réel ou non. Cette distinction entre les tweets relatifs aux désastres et ceux qui ne le sont pas, bien que semblant simple en surface, présente plusieurs défis importants liés à l'ambiguïté du langage et à la variabilité des expressions humaines. Sur une plateforme de réseau social, il est fréquent de rencontrer de l'ironie, des fausses informations, et d'autres subtilités. Ces complexités m'ont motivé à utiliser des modèles d'apprentissage profond comme BERT et BERTweet.

1.2 Données

Dans cette compétition, nous avons deux jeux de données au format CSV contenant les tweets. Le premier, un jeu de données d'apprentissage annoté, compte 7503 entrées avec les colonnes `<text>`, `<target>`, `<keyword>`, `<location>`. Le deuxième jeu de test, non annoté, contient 3243 entrées. Les résultats des prédictions sur ce jeu de données test sont évalués en les fusionnant avec le fichier `sample_submission.csv`, qui inclut une colonne `<id>` pour l'identifiant du tweet et `<target>` pour le label (0 ou 1). Après la soumission de ces prédictions, un score est attribué et les participants sont classés dans la compétition. Tous les tweets sont en anglais et présentent une variété en termes de structure et de style, reflétant ainsi la diversité naturelle des communications sur les réseaux sociaux.

Cette richesse de données offre des avantages mais pose également des défis pour l'analyse éventuelle :

- **Volume et variété :** La quantité de données disponible permet de former des modèles robustes capables de généraliser à partir de nombreux exemples. Cependant, la variabilité des expressions, incluant les argots, les abréviations et les erreurs orthographiques, complique le processus de modélisation.
- **Les biais :** En tant que données générées par les utilisateurs, les tweets peuvent contenir des informations incorrectes ou être sujettes à des biais.

Ces facteurs rendent la tâche de classification particulièrement complexe et motivent l'utilisation de modèles avancés de traitement automatique du langage naturel.

1.3 Problématique

Outre la problématique principale de ce projet, qui consiste à détecter les tweets liés aux désastres, il était essentiel pour moi de comprendre comment ces prédictions allaient être effectuées et surtout, d'analyser l'impact des différentes techniques et approches d'apprentissage sur cette même question.

Les questions que je me suis posées sont les suivantes :

1. Comment peut-on améliorer la classification des tweets en utilisant des modèles de traitement du langage naturel (NLP) avancés comme BERT et BERTweet ?
2. Est-il possible d'atteindre ou de surpasser les performances d'un modèle préentraîné spécifiquement pour les tweets (BERTweet) avec un modèle généraliste comme BERT-base ?

Pour explorer ces questions, j'ai opté pour une technique de préentraînement de masquage, ou MLM (Masked Language Modeling), afin de tenter d'améliorer les performances du modèle BERT-base et de les comparer à celles de BERTweet.

1.3.1 Hypothèses

— **Utilisation du pré-entraînement avec masquage (MLM) :** Cette technique pourrait enrichir l'espace représentatif des caractéristiques (features) et, par conséquent, améliorer les performances du modèle BERT-base.
— **Performance du modèle pré-entraîné :** Un modèle spécifiquement préentraîné sur un grand volume de tweets (BERTweet) devrait logiquement atteindre de meilleures performances que BERT-base dans ce contexte spécifique.

Ces hypothèses visent à comparer les performances des deux modèles dans la classification des tweets relatifs aux désastres, offrant ainsi une perspective comparative entre une approche généraliste et une approche spécialisée.

1.4 Objectifs

L'objectif principal de ce projet est de comparer les différentes techniques et approches d'apprentissage pour évaluer leur impact sur la tâche de classification des tweets. Pour cela, il est nécessaire d'analyser les phases de pré-entraînement, de transfert d'apprentissage et de fine-tuning :

— **Évaluer et comparer les performances de BERT et BERTweet sur la tâche de classification des tweets :** Cette comparaison permettra de déterminer si un modèle général comme BERT peut atteindre des performances similaires à celles d'un modèle spécialisé comme BERTweet.
— **Analyser l'impact du pré-entraînement avec masquage sur les résultats :** Il est nécessaire de comparer le modèle avec un pré-entraînement et sans.
— **Explorer différentes configurations d'hyperparamètres pour optimiser le modèle :** L'ajustement des hyperparamètres est crucial pour maximiser la performance du modèle.

Afin de garantir une évaluation dans des conditions les plus équitables possibles, les mêmes jeux de données d'entraînement et les mêmes configurations d'hyperparamètres seront utilisés pour les différents modèles. Pour évaluer et comparer les résultats de ces modèles, nous utiliserons les prédictions soumises sur la plateforme Kaggle.

Ce projet m'a permis d'atteindre mon objectif personnel d'exploration et d'application de différentes techniques, en étudiant leur impact et en enrichissant mon expérience . Il m'a également offert l'opportunité de m'entraîner à la prise de décisions stratégiques liées aux problèmes de machine learning. Cette exploration est essentielle pour développer des compétences avancées en traitement automatique du langage naturel et pour me préparer à des projets futurs dans ce domaine en pleine expansion.

Chapitre 2

Méthodologie

2.1 Outils et technologies utilisés

FIGURE 2.1 – Logo Hugging Face.

FIGURE 2.2 – Logo PyTorch.

Pour le développement de ce projet, j'ai utilisé les outils et technologies suivants :

— **Hugging Face Transformers [3] :** Une bibliothèque populaire en Python qui fournit des modèles Transformers tels que BERT et BERTweet. Leur site propose également des tutoriels qui m'ont aidé à développer mon code.

— **Trainer [4] :** Une classe de la bibliothèque Hugging Face Transformers qui facilite les phases d'apprentissage et l'évaluation des modèles. Cette bibliothèque permet également de sauvegarder facilement les modèles après l'entraînement.

— **PyTorch [5] :** La bibliothèque principale pour le développement du code lié aux modèles.

— **scikit-learn [6] :** Une bibliothèque de machine learning en Python, utilisée pour les tâches de prétraitement des données et d'évaluation des modèles.

Gestion de projet

FIGURE 2.3 – Logo Jupyter, Google Colab.

FIGURE 2.4 – Logo Notion.

Pour gérer une grande quantité de code testé et des modèles développés, j'ai utilisé **Google Colab** pour développer dans des notebooks Jupyter et **Google Drive** pour sauvegarder mes avancées, ainsi que les jeux de données et les modèles entraînés pour une utilisation ultérieure.

Au début du deuxième semestre, j'ai remarqué que souvent je ne comprenais pas profondément mon code. Cependant, les exemples et les tutoriels offerts par Hugging Face ont permis d'accélérer ce travail et de développer une compréhension globale du développement, du but et des nuances de chaque étape. Le développement des modèles d'apprentissage prend beaucoup de temps avec des ressources limitées. Mais l'accès aux GPU offert par Google Colab, la possibilité de sauvegarder le modèle à chaque étape du développement, et l'utilisation de modèles BERT plus "légers" limitent cet effet.

Par exemple, après le préentraînement par masquage, je sauvegarde mon modèle et je le reprends ultérieurement avec les poids et les configurations sauvegardés pour l'utiliser dans les tâches en aval.

De plus, avec le rythme du master, il était essentiel de reprendre le travail du dernier point sans perdre les idées, les performances des modèles, les avancées et les tâches à effectuer. Pour cela, j'ai utilisé l'application **Notion** qui permet de créer des notes avec des fonctionnalités très flexibles. Cette application est également utilisée dans des entreprises et des startups pour la gestion de projet. En utilisant ces outils, j'ai pu maintenir une organisation efficace et assurer la continuité du travail malgré les contraintes de temps et de ressources.

2.1.1 BERT-base

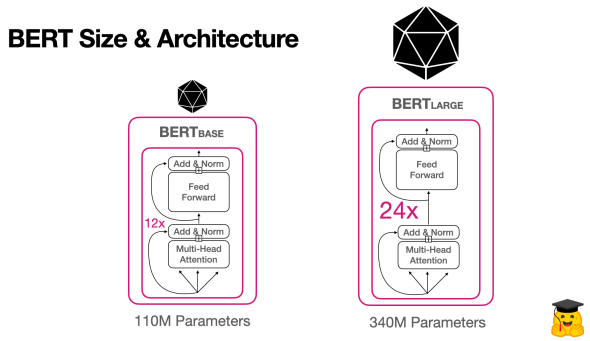

FIGURE 2.5 – BERT-base et BERT-large

BERT (Bidirectional Encoder Representations from Transformers) [7] est un modèle pré-entraîné sur une grande quantité de texte provenant de Wikipédia et du BooksCorpus, comprenant environ 110 millions de paramètres. BERT-base est conçu pour être plus léger que sa variante BERT-large, ce qui le rend plus accessible pour des projets avec des ressources limitées.

Pour adapter BERT-base aux tâches spécifiques de ce projet, deux étapes principales sont suivies. Premièrement, je procède au pré-entraînement du modèle en utilisant la technique du BERT for MLM (Masked Language Model), qui constitue la dernière couche conçue spécifiquement pour ce type de pré-entraînement. Deuxièmement, cette dernière couche est remplacée par une couche de classification, tout en conservant les mêmes poids et paramètres pour continuer avec l'ajustement du modèle.

2.1.2 BERTweet-base

BERTweet [8] est une variante de BERT spécialement pré-entraînée sur une grande quantité de données issues de Twitter, incluant des millions de tweets. Ce modèle contient environ 135 millions de paramètres. BERTweet-base est particulièrement adapté pour comprendre le langage et le contexte des tweets.

La sélection de BERTweet-base pour ce projet repose sur son pré-entraînement spécifique sur les données des tweets, offrant théoriquement de meilleures performances pour la classification des tweets par rapport à un modèle généraliste comme BERT-base.

2.2 Pré-traitement des données

Bien que les données soient déjà adaptées à une utilisation avec des modèles comme BERT, qui ne nécessitent pas de traitement préliminaire tel que la lemmatisation ou la suppression des stopwords, il était essentiel de garantir une bonne qualité des données avant de procéder à l'apprentissage sur nos tâches.

Tout d'abord, j'ai procédé à la suppression des liens et des emails, qui seraient inutiles pour BERT et BERTweet. Les séquences telles que les hashtags, les mentions et les emojis sont interprétables par BERTweet, c'est pourquoi j'ai décidé de les conserver.

Notre jeu de données d'entraînement contenait des doublons, parfois avec des labels différents, ce qui pouvait biaiser les résultats de l'apprentissage. Après avoir examiné certains de ces doublons, j'ai décidé de supprimer tous les tweets avec des labels en conflit pour éviter toute ambiguïté.

Puisque j'allais effectuer plusieurs tests durant tout le semestre, il était important de garder les mêmes structures des jeux de données. Pour cela, j'ai utilisé le shuffle et son random state pour pouvoir réutiliser le même tirage ultérieurement. J'ai ensuite divisé le jeu de données en deux sous-ensembles pour le fine-tuning : un ensemble "train" et un ensemble de "validation", dans un ratio de 80-20% respectivement.

2.2.1 Tokenization

Cette étape implique de décomposer le texte en caractères, appelés tokens. La tokenization permet de convertir les mots en données numériques et de les représenter dans des espaces multidimensionnels. Puisque BERT est déjà préentraîné sur un grand corpus, il sait déjà où le vecteur du mot est placé dans l'espace pour former les "word embeddingds".

Dans ce processus, plusieurs listes de caractères sont générées pour rendre les données interprétables au modèle [9] :

— **les input ids :** Chaque token est associé à un identifiant unique selon un vocabulaire prédéfini. Ce vocabulaire est construit lors de la phase de pré-entraînement, ces ids sont ensuite utilisés pour retrouver les embeddings correspondants.

— **attention mask :** Une liste pour différencier les tokens qui forment le texte, des tokens de remplissage comme [PAD].

— **labels :** Liste des labels associés.

Le token [CLS] est principalement responsable de la tâche de classification et s'ajoute automatiquement au début du texte lors de l'utilisation de la librairie transformers. Il agrège ensuite les embeddings à travers toute la séquence d'entrée grâce au mécanisme d'auto-attention. Cela résulte en une représentation sous forme de vecteur qui représente les relations contextuelles au sein du texte. Ce vecteur est ensuite introduit dans le classifieur pour déterminer la classe à laquelle appartient le texte.

Pour BERT, toutes les séquences d'entrée doivent être uniformisées en terme de longueur, ce qui se fait automatiquement durant la phase de tokenization en ajoutant des tokens [PAD] jusqu'à ce que la longueur fixée soit atteinte. La longueur maximale autorisée est de 512 tokens, mais étant donné que les tweets sont généralement courts, nous avons déterminé la longueur maximale en fonction de la longueur du tweet le plus long dans notre jeu de données.

2.3 Analyse descriptive

Avant de procéder à l'apprentissage, il est important de vérifier que les données sont prêtes et adaptées pour l'entraînement.

J'ai d'abord analysé la répartition des labels des tweets. Cette étape permet de détecter une éventuelle disparité entre les classes, ce qui pourrait nécessiter d'ajuster les méthodes d'apprentissage et les fonctions de coût afin que le modèle puisse apprendre efficacement à classer des tweets qui sont relativement "rares". Dans notre cas, bien que la distribution ne soit pas parfaitement équilibrée avec 4090 tweets annotés comme '0' (non-désastre) et 2822 comme '1' (désastre), j'ai estimé que le volume était suffisant pour poursuivre sans nécessiter d'ajustements majeurs.

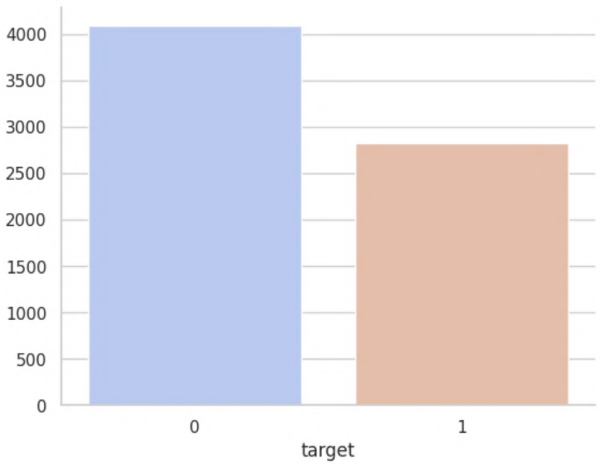

FIGURE 2.6 – Distribution des labels

J'ai ensuite examiné la distribution des longueurs des tweets pour déterminer la longueur maximale de tokens nécessaire. Cette démarche vise à garantir que le modèle traite les données de manière efficace, en minimisant l'ajout de padding inutile qui pourrait rallonger inutilement le temps de traitement.

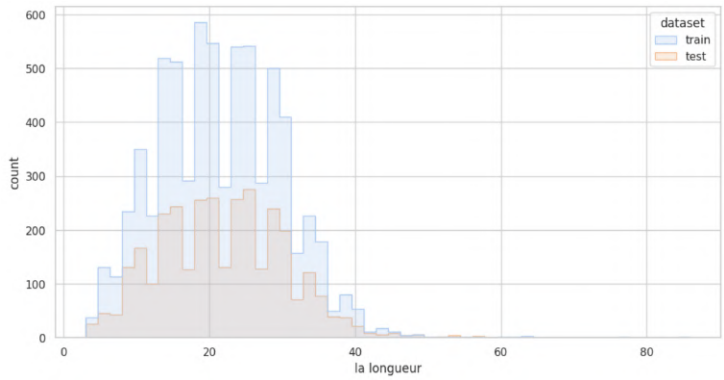

FIGURE 2.7 – Distribution de la longueur des tweets

Finalement, j'ai réalisé une analyse des mots les plus courants dans les tweets ayant le label '1', afin de confirmer que ces tweets sont effectivement liés à des désastres.

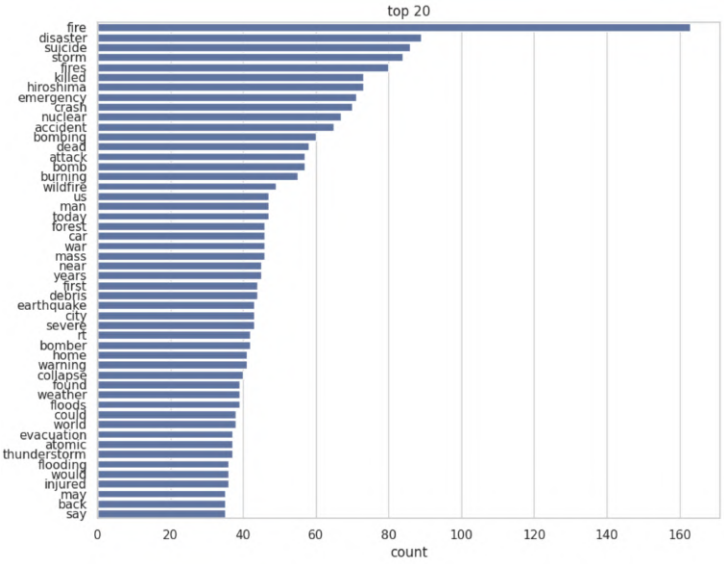

FIGURE 2.8 – top-20 mots les plus courants dans les tweets désastres

2.4 Transfer Learning et le Masked Language Modeling (MLM)

Au début du second semestre, un apprentissage complémentaire à l'apprentissage classique a été proposé par mon tuteur universitaire afin d'enrichir l'espace représentatif du modèle préentraîné. Le préentraînement de masquage (Masked Language Modeling [10]) permettrait au modèle d'avoir une compréhension plus profonde des textes cibles, ce qui pourrait potentiellement avoir un impact positif en termes de performances.

Dans cette phase complémentaire, j'utilise `BertForMaskedLM`, une couche spécifique conçue pour le préentraînement par masquage, que j'ajoute au modèle BERT-base. Cette approche consiste à masquer aléatoirement des mots dans les phrases, et le modèle doit ensuite prédire ces mots manquants en se basant sur les mots et le contexte du texte. Implicitement, nous faisons apprendre au modèle ce qu'est un tweet, en termes de langage et de style. Après cette étape, je devais effectuer le "transfert" de l'apprentissage à l'étape de fine-tuning en changeant cette dernière couche.

J'ai donc décidé d'ajuster le modèle BERT-base aux spécificités et au style du langage utilisé dans les tweets. Pour cela, j'ai utilisé un jeu de données trouvé sur Kaggle, intitulé "Disaster Tweets" [11], qui contient environ 11 000 textes annotés de la même manière que le jeu de données initial, avec des tweets liés aux désastres et d'autres qui ne le sont pas. Ce jeu de données, créé par un utilisateur, est destiné à enrichir le dataset que nous utilisons déjà dans notre tâche.

Le masquage est réalisé lors de la phase de tokenisation, où j'ai introduit un taux de masquage de 30% du texte total. Pour optimiser le processus, j'ai également créé une liste de mots (*stop words*) [12] courants en anglais. Cela permet d'éviter le masquage de mots ou de caractères peu significatifs et encourage le modèle à effectuer un apprentissage plus efficace.

Cette étape a été particulièrement complexe, car je devais cibler uniquement les tokens éligibles au masquage. Au départ, je masquais souvent des tokens spéciaux comme [CLS] et [PAD], ce qui aurait été incorrect pour l'apprentissage. Pour cela, je parcours un par un tous les tokens et vérifie qu'ils ne sont ni dans la liste d'exclusion ni parmi les tokens spéciaux, tout en sachant qu'il est nécessaire de masquer 30% du texte. Je devais donc tenir compte des cas où il y a plus de tokens à masquer que de tokens éligibles. Dans ce cas, je vais masquer seulement les tokens éligibles, même si cela représente moins de 30% des tokens à masquer.

Après la tokenisation des nouveaux tweets et leur division en ensembles de train et de validation avec un ratio de 70-30%, je passe à la phase de préentraînement utilisant quatre époques et un batch size de 32. Cette étape suit la même méthodologie que celle du fine-tuning, à la différence près que l'objectif est de prédire le mot masqué plutôt que le label du texte. Le modèle essaie donc de prédire le mot caché et, en cas d'échec, il est pénalisé, ce qui l'oblige à ajuster ses poids en conséquence.

Une fois ce pré-entraînement complété, je sauvegarde le modèle dans mon Google Drive. Je le recharge ultérieurement pour changer la dernière couche en une couche de classification : `BertForSequenceClassification`. Cette approche me permet de réutiliser le modèle ajusté pour la tâche spécifique de classification binaire des tweets.

2.5 Fine-tuning

Le fine-tuning est la phase clé où le modèle est ajusté pour la tâche de classification binaire, dans le but d'obtenir une bonne généralisation sur des données jamais vues auparavant. Une compréhension approfondie des notions liées au processus d'apprentissage et à ces résultats est cruciale, car elle permet de prendre des décisions stratégiques qui orientent le modèle vers les meilleures performances sur une tâche spécifique.

2.5.1 Hyperparamètres

Trois hyperparamètres clés définissent le déroulement de l'apprentissage :

Époques : Pour la phase de fine-tuning, j'ai fixé à 6 le nombre d'époques, durant lesquelles le meilleur modèle est choisi en terme d'accuracy pour être sauvegardé à la fin de l'entraînement. Cette décision est justifiée dans les cas où notre modèle présente une augmentation de la loss de validation tout en affichant une accuracy croissante. Ainsi, je sauvegarde l'époque où l'accuracy atteint son maximum.

Taille du lot (Batch size) : Après quelques tests sur les modèles BERT-base et BERT-Tweet, j'ai empiriquement fixé la taille de lot à 32. Cette taille a montré une meilleure convergence lors de la phase de validation et a été, par conséquent, adoptée pour les tests.

Taux d'apprentissage (Learning rate) : En consultant la documentation de BERT, qui recommande un taux d'apprentissage compris entre 2e-5 et 5e-5, j'ai opté, après quelques essais, pour un taux plus faible de 1e-5. Ce choix s'est également avéré efficace pour obtenir une bonne convergence.

2.5.2 Optimisation du modèle

L'optimisation du modèle est réalisée avec **AdamW** [13], un optimiseur choisi par défaut lors de l'utilisation de la bibliothèque transformers. Il est crucial de comprendre comment cet algorithme ajuste les paramètres et les poids pour pouvoir interpréter les résultats ultérieurs et la sélection des hyperparamètres.

Algorithm 2 Adam with L_2 regularization and Adam with decoupled weight decay (AdamW)

given $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, \lambda \in \mathbb{R}$
initialize time step $t \leftarrow 0$, parameter vector $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \boldsymbol{0}$, second moment vector $\boldsymbol{v}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
repeat
  $t \leftarrow t + 1$
  $\nabla f_t(\boldsymbol{\theta}_{t-1}) \leftarrow \text{SelectBatch}(\boldsymbol{\theta}_{t-1})$ ▷ select batch and return the corresponding gradient
  $\boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{\theta}_{t-1}) + \lambda \boldsymbol{\theta}_{t-1}$
  $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t$ ▷ here and below all operations are element-wise
  $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t^2$
  $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1 - \beta_1^t)$ ▷ β_1 is taken to the power of t
  $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t / (1 - \beta_2^t)$ ▷ β_2 is taken to the power of t
  $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$ ▷ can be fixed, decay, or also be used for warm restarts
  $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta_t \left(\alpha \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon) + \lambda \boldsymbol{\theta}_{t-1} \right)$
until *stopping criterion is met*
return optimized parameters $\boldsymbol{\theta}_t$

FIGURE 2.9 – Algorithme de Adam et AdamW

AdamW est une adaptation de l'optimiseur Adam avec le weight-decay. Lorsque Adam est appliqué sur une fonction de perte f plus une régularisation L2, les poids qui ont tendance à avoir de grands gradients dans f ne sont pas autant régularisés qu'ils le seraient avec un weight decay découplé, puisque le gradient du régularisateur est également affecté par l'échelle des gradients de f. En revanche, dans notre cas, le weight decay découplé régularise tous les poids au même taux λ , permettant ainsi de régulariser de manière implicite le modèle en imposant une pénalité plus importante aux poids plus élevés, ce qui conduit à une meilleure généralisation que la régularisation L2. AdamW attribue un taux d'apprentissage individuel à chaque paramètre, prenant des pas plus petits là où la pente de la fonction de perte est abrupte, et des pas plus grands là où le gradient est plus plat. Cette dissociation empêche également la réduction des taux d'apprentissage qui se produit avec Adam traditionnel lorsque le weight decay est inclus.

Warm-up steps :
Pour cet optimiseur, un nombre de "warm-up steps" est également fixé, ce qui spécifie le nombre d'itérations au début de l'entraînement durant lesquelles le taux d'apprentissage est progressivement accéléré. Cette technique, partant de presque zéro jusqu'à la valeur initialement fixée, permet d'atteindre une convergence plus stable et progressive.

2.5.3 Fonction de coût

La fonction de coût utilisée est la **Cross-Entropy Loss**. Cette fonction est appliquée après la couche de softmax, qui convertit les scores prédits du modèle en probabilités réelles. Elle compare la distribution de probabilités prédites par le modèle pour chaque classe avec la distribution de probabilités réelle, pénalisant les prédictions incorrectes, particulièrement celles qui sont faites avec une grande confiance.

2.6 Critères d'évaluation

Lors de l'évaluation des prédictions sur les ensembles de validation, j'ai utilisé les critères classiques tels que l'accuracy, le F1-score, la précision et le rappel. Au début de mes développements, lorsque je recherchais les hyperparamètres optimaux pour l'apprentissage, je me suis basé sur les performances de l'accuracy et du F1-score, tout en tenant compte de l'évolution de la loss de validation à travers les époques. Idéalement, je cherchais des performances où les pertes de validation et d'entraînement diminuaient graduellement jusqu'à ce que l'accuracy atteigne son maximum. Cependant, j'ai parfois constaté lors de tests que même avec une augmentation de la loss de validation, l'accuracy augmentait également, ce qui était pour moi acceptable pour juger que l'apprentissage était correct.

Ainsi, pour les soumissions de prédictions sur la plateforme Kaggle, les performances sont évaluées en utilisant le score F1, défini comme suit :

$$F1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

où la précision (*precision*) et le rappel (*recall*) sont calculés par :

$$\text{precision} = \frac{TP}{TP + FP}$$

$$\text{recall} = \frac{TP}{TP + FN}$$

Chapitre 3

Résultats et Discussions

3.1 Résultats sans MLM

Pour évaluer les performances sans la phase de pré-entraînement complémentaire, j'ai réalisé des tests avec BERTweet et BERT-base en utilisant le fine-tuning classique. Comme mentionné précédemment, j'ai utilisé les mêmes jeux de données et les mêmes configurations d'hyperparamètres : Batch size de 32, Learning rate de 1e-5, warmup_steps de 50, weight_decay de 0.02.

Résultats de BERT-base : Lors des prédictions, BERT-base a atteint une accuracy maximale de 0.833695 avec un F1-score de 0.776699 à la troisième époque (Figure A.1). Après cette époque, j'ai observé une diminution de l'accuracy avec une augmentation graduelle de la loss de validation.

Résultats de BERTweet-base : BERTweet, quant à lui, a atteint une accuracy maximale de 0.850295 avec un F1-score de 0.809045 (Figure A.2).

Comparaison et Analyse

Ensuite, j'ai effectué des prédictions sur le jeu de données de test pour ensuite examiner les scores attribués sur Kaggle

— BERT-base a atteint un score de 0.83266.

— BERTweet a atteint un score de 0.83757.

Malgré une petite différence de score, on peut constater que BERTweet a légèrement mieux performé par rapport à BERT-base, ce qui était logique vu la nature du pré-entraînement de BERTweet.

Ces résultats m'ont permis de supposer que si la méthode de masquage est efficace, les performances du modèle avec masquage devraient se situer entre celles des deux autres modèles.

3.2 Résultats avec MLM

Après avoir rechargé le modèle BERT-base pré-entraîné sur la tâche de masquage, j'ai procédé au fine-tuning de la même manière.

J'ai constaté une accuracy maximale de 0.843095 avec un F1-score de 0.797386 (Figure A.5). Les époques suivantes ont également montré une diminution en accuracy et une augmentation de la loss de validation.

Comparaison et Analyse :

TABLE 3.1 – Scores Kaggle des différents modèles

Modèle	**Score Kaggle**
BERT-base	0.83266
BERT-base avec MLM	0.83542
BERTweet	0.83757

Concernant les scores attribués aux prédictions sur Kaggle, le modèle a atteint un score de 0.83542. Cette performance était parfaitement en cohérence avec mes attentes, confirmant l'efficacité de l'intégration de la phase de MLM.

3.3 Réduction de la taille de lot

Cependant, l'obtention de résultats si similaires après ces trois tests utilisant des modèles différents a attiré mon attention, même si l'ordre des performances correspond à mes attentes initiales.

Par conséquent, j'ai voulu tester une hypothèse qui m'est venue à l'esprit : utiliser une taille de lot beaucoup plus petite permettrait d'atteindre une meilleure généralisation, en raison des mises à jour plus fréquentes des poids et des paramètres, ce qui conduirait à une pénalisation plus sévère du modèle.

Ainsi, j'ai repris mon modèle pré-entraîné avec la phase de masquage, et j'ai procédé au fine-tuning. Malgré l'instabilité observée avec de petits batch sizes, notamment une augmentation de la loss de training dans certaines époques, cette fois-ci, un batch size de 8 a été fixé, avec le même learning rate de 1e-5, mais avec un warmup_steps de 100 et un weight_decay de 0.01.

Le modèle a présenté une loss de validation constamment croissante, mais a atteint son accuracy maximale lors de la deuxième époque avec une accuracy de 0.849602 et un F1-score de 0.809174 (Figure A.6).

Comparaison et Analyses

Ensuite, sur les prédictions soumises sur Kaggle, le modèle a réussi à atteindre un score de 0.84308, ce qui était bien meilleur que tous les résultats précédents. Ce résultat peut probablement s'expliquer par le fait que la réduction de la taille du lot "aplatit" la surface de la fonction de perte, permettant ainsi au modèle de sortir des minimums locaux atteints avec une taille de lot de 32 et de trouver une solution plus optimale contribuant à une meilleure généralisation.

TABLE 3.2 – Scores Kaggle des différents modèles *bs – batch size

Modèle	**Score Kaggle**
BERT-base	0.83266
BERT+MLM	0.83542
BERTweet	0.83757
BERT+MLM bs : 8	0.84308

3.3.1 Extension des tests à d'autres modèles

Ces résultats m'ont encouragé à effectuer le même test pour d'autres modèles afin de comparer globalement cet effet. Pour cela, j'ai réutilisé les mêmes hyperparamètres.

— **BERT-base** a atteint une accuracy maximale de 0.827187 avec un F1-score de 0.789798 à la deuxième époque (Figure A.3).
— **BERTweet** a atteint aussi une accuracy maximale de 0.827187 avec un F1-score de 0.777674 à la deuxième époque (Figure A.4).

Pour les prédictions sur Kaggle :
— BERT-base a atteint un score de 0.83144.
— BERTweet a atteint un score de 0.83604.

Ces deux modèles ont montré des performances inférieures par rapport à celles obtenues avec un batch size de 32. J'ai donc réalisé un deuxième test pour le modèle avec le pré-entraînement MLM, pour m'assurer que le résultat obtenu n'était pas aléatoire. Pour cela, j'ai répété le pré-entraînement de masquage et le fine tuning. Le deuxième score obtenu est de 0.83941, ce qui est toujours plus élevé que tous les scores obtenus par les autres modèles.

Analyse des résultats :

La raison exacte pour laquelle les modèles n'ont pas réussi à mieux généraliser avec une taille de lot plus petite reste inexplorée. Peut-être que les modèles sans pré-entraînement de masquage tombent rapidement dans une situation d'overfitting et n'arrivent pas à

généraliser sur de nouveaux exemples. Bien que le modèle BERTweet soit spécifiquement conçu pour le contexte des tweets, il reste tout de même plus général que BERT-base avec pré-entraînement de masquage. Dans le nouveau jeu de données sur lequel BERT-base a été pré-entraîné, il y avait une répartition plus ou moins équilibrée des exemples de tweets liés aux désastres et ceux qui ne le sont pas, ce qui a peut-être permis de mieux se préparer à cette spécificité. En réduisant la taille du lot, le modèle a peut-être atteint une meilleure généralisation.

3.4 Analyse t-SNE

Il était également intéressant d'observer comment les modèles parviennent à séparer les textes en clusters de classes dans un espace de grande dimension.

Une méthode utile pour cette analyse est la projection t-SNE, qui permet de réduire la dimensionnalité des données de haute dimension pour une visualisation en deux ou trois dimensions.

L'interprétation des projections de données de haute dimension en deux ou trois dimensions est souvent très subtile et dépend beaucoup de la technique utilisée. C'est également le cas avec la t-SNE [14]. Afin de projeter, je dois fixer deux paramètres qui influenceront les projections réalisées :

— La perplexité
— Le taux d'apprentissage (learning rate)

En ajustant ces paramètres, on peut changer significativement la projection des points sur la surface. Il est souvent recommandé d'utiliser une perplexité entre 5 et 50 et un taux d'apprentissage assez élevé pour mieux agréger les points. Pour cela, j'ai utilisé les derniers modèles fine-tunés avec un batch size de 8, et projeté la séparation des classes sur la base de données d'entraînement. J'ai fixé la perplexité à 50 et le learning rate à 1000 pour les projections de mes trois modèles.

Modèle avec MLM :

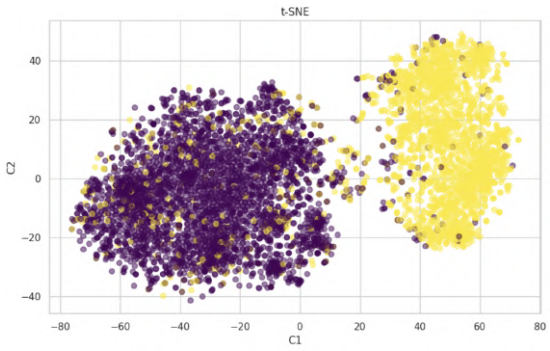

FIGURE 3.1 – t-SNE modèle avec MLM

La visualisation montre deux groupes bien distincts, sauf quelque points qui se forment entre les deux clusters. Cette séparation nette peut signifier que le pré-entraînement avec masquage aide le modèle à mieux comprendre les variations dans les données, ce qui se traduit par une meilleure séparation dans l'espace de représentation.

Modèle sans MLM :

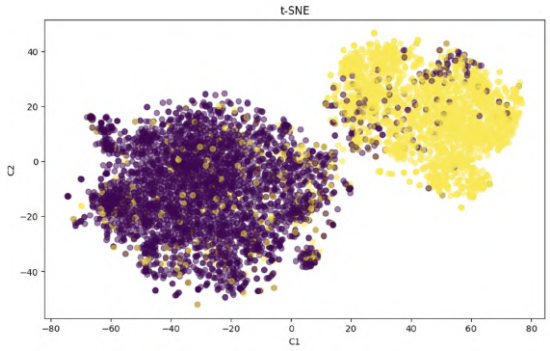

FIGURE 3.2 – t-SNE modèle sans MLM

Dans cette visualisation, bien que deux groupes principaux soient encore visibles, ils semblent moins distincts comparés au modèle avec MLM. Il y a plus de chevauchement dans la classe 1 (points jaunes) Cela pourrait suggérer que sans l'étape de pré-entraînement de masquage, le modèle a une capacité légèrement réduite à différencier les catégories ou les types de données.

Modèle BERTweet

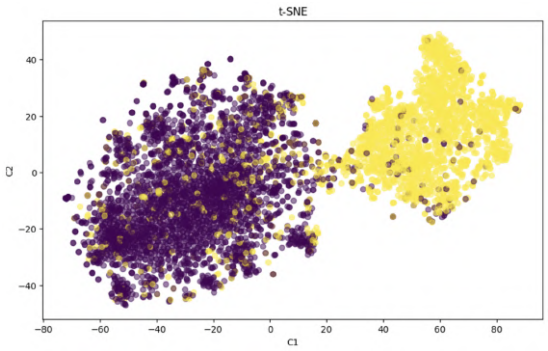

FIGURE 3.3 – t-SNE BERTTweet

BERTweet montre également deux clusters principaux, mais avec un chevauchement différent de celui observé dans le modèle sans MLM. On peut voir aussi des formes des petits clusters surtout autour de la classe 0 (les points violets). Étant donné que BERTweet est optimisé pour les tweets, cette disposition pourrait refléter des particularités ou des caractéristiques spécifiques des données liées au langage des tweets, offrant une bonne séparation mais avec un caractère distinct comparé au modèle générique pré-entraîné avec MLM.

3.5 Validation croisée

Ayant utilisé initialement l'approche train-validation, j'ai ensuite voulu observer les performances du modèle le plus performant, configuré avec un batch size de 8, à travers l'ensemble du jeu de données.Pour cela, j'ai employé une validation croisée en 5 folds qui couvre la totalité du jeu de données.

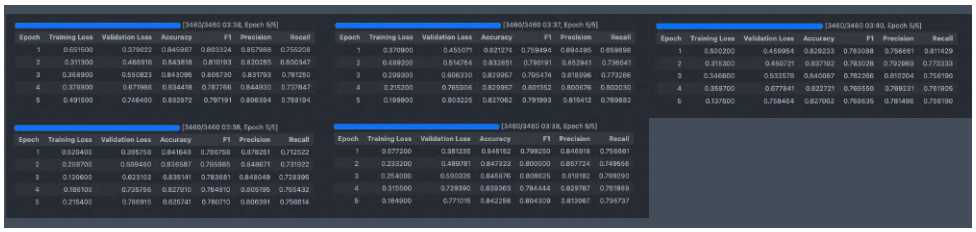

FIGURE 3.4 – Les folds de la validation croisée

J'ai constaté que le modèle est assez stable sur les cinq folds en termes d'accuracy et de score F1. Toutefois, le troisième fold a montré une légère fluctuation de l'accuracy, se situant autour de 0.82-0.83, tandis que pour les autres folds, elle se maintenait plutôt autour de 0.84. Globalement, le modèle montre une bonne performance stable, ce qui confirme que ce n'étaient pas uniquement des points « faciles à classer » qui ont obtenu de bonnes performances lors des tests précédents.

Chapitre 4

Conclusion et Perspectives

Les analyses réalisées ont permis de mettre en évidence l'efficacité des méthodes de fine-tuning et de pré-entraînement avec masquage des mots pour améliorer la performance des modèles BERT sur des tâches de classification de texte, et même d'atteindre de meilleurs résultats que le modèle spécialisé sur les tweets. Ce projet démontre également l'importance de la maîtrise de l'optimisation des hyperparamètres et de la sélection des modèles dans les problèmes de machine learning. L'utilisation de la validation croisée a également démontré la stabilité du modèle à travers les différents sous-ensembles de données, confirmant ainsi la bonne sélection du modèle. Ces découvertes répondent directement à la problématique initiale visant à comprendre comment différentes configurations et méthodes d'entraînement influencent la performance des modèles de traitement du langage naturel.

4.1 Compétences acquises

Au cours de ce projet, j'ai développé plusieurs compétences techniques et analytiques. Parmi celles-ci, la maîtrise des techniques de machine learning avancées, notamment pour les modèles de traitement du langage naturel comme BERT et BERTweet. Cette expérience m'a également permis de renforcer ma compréhension de la théorie derrière l'apprentissage profond, en offrant une pratique constante et la possibilité d'appliquer ces notions tout au long de l'année.

4.2 Adéquation avec mes études

Ce projet m'a offert l'opportunité d'appliquer mes connaissances théoriques et de les approfondir en développant le déroulement complet d'une tâche prédictive. Avec du recul, je réalise comment le projet a évolué au fil du temps et comment les cours de machine learning et de deep learning m'ont aidé à progresser rapidement.

4.3 Difficultés rencontrées

Personnellement, je trouve que la plus grande difficulté était de m'adapter au rythme de travail requis tout en comprenant parallèlement les notions appliquées. Sachant qu'au début de l'année, je ne savais pas ce qu'était un apprentissage supervisé, il était assez difficile de comprendre les aspects qui demandent beaucoup d'attention en pratique. Parmi les difficultés techniques, stabiliser les performances du modèle à travers différentes configurations et prévenir l'overfitting ont été particulièrement complexes. L'ajustement des hyperparamètres, tels que la taille du lot et le taux d'apprentissage, a nécessité une série d'expérimentations méthodiques et parfois répétitives.

4.4 Perspectives

Bien que le projet ait montré des résultats assez intéressants, je pense qu'il n'y a pas de limites aux solutions et améliorations possibles, non seulement pour cette tâche spécifique mais aussi plus largement. Cela prouve à la fois la puissance et la complexité des techniques et des notions appliquées. Par exemple, une phase de sélection des hyperparamètres plus approfondie et méthodique serait bénéfique. De plus, une exploration plus poussée du préentraînement par masquage, y compris avec différentes variantes et taux de masquage, serait instructive. Les modifications de la fonction de perte auraient également été intéressantes à tester. Un préentraînement à la prédiction des phrases suivantes est aussi une idée à explorer. Et ce ne sont que quelques-unes des nombreuses possibilités à appliquer pour ce projet.

Mon objectif principal pour l'avenir est de pouvoir appliquer ces nouvelles découvertes précieuses dans le monde professionnel. Que ce soit en alternance ou dans un emploi principal, je vais continuellement essayer de mettre en œuvre des solutions de machine learning et de deep learning grâce à la puissance de ces notions scientifiques qui démontrent déjà aujourd'hui comment notre société évolue avec ces techniques.

Bibliographie et Ressources

[1] Ashish Vaswani, Llion Jones, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf, 2017.

[2] Phil Culliton Yufeng Guo Addison Howard, devrishi. Natural language processing with disaster tweets. https://kaggle.com/competitions/nlp-getting-started, 2019.

[3] Hugging face. https://huggingface.co.

[4] Hugging Face Trainer. https://huggingface.co/docs/transformers/v4.41.2/en/main_classes/trainer#transformers.Trainer.

[5] Pytorch. https://pytorch.org.

[6] scikit-learn. https://scikit-learn.org/stable/.

[7] Kenton Lee Kristina Toutanova Jacob Devlin, Ming-Wei Chang. Bert : Pre-training of deep bidirectional transformers for language understanding, 2019.

[8] Dat Quoc Nguyen, Thanh Vu, and Anh Tuan Nguyen. Bertweet : A pre-trained language model for english tweets. EMNLP 2020 Demos, 2020. VinAI Research, Vietnam ; Oracle Digital Assistant, Oracle, Australia ; NVIDIA, USA.

[9] Hugging Face : Fine tuning with custom datasets. https://huggingface.co/transformers/v3.1.0/custom_datasets.html#seq-imdb.

[10] Hugging Face : Fine tuning a masked language model. https://huggingface.co/learn/nlp-course/en/chapter7/3.

[11] Viktor Stepanenko and Iryna Liubko. Disaster tweets. https://www.kaggle.com/dsv/1640141, 2020.

[12] English stop words. https://gist.github.com/sebleier/554280.

[13] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. arXiv preprint arXiv :1711.05101, 2017.

[14] Martin Wattenberg, Fernanda Viégas, and Ian Johnson. How to use t-sne effectively. http://distill.pub/2016/misread-tsne, 2016.

Annexe A

Détails Supplémentaires

A.1 Détails des résultats sans MLM

[1038/1038 03:14, Epoch 6/6]

Epoch	Training Loss	Validation Loss	Accuracy	F1	Precision	Recall
1	0.390900	0.412947	0.832972	0.781870	0.838057	0.732743
2	0.382500	0.439853	0.817787	0.784615	0.758678	0.812389
3	0.332200	0.406970	0.833695	0.776699	0.860215	0.707965
4	0.238100	0.459796	0.823572	0.780576	0.793419	0.768142
5	0.249000	0.472223	0.823572	0.774492	0.810445	0.741593
6	0.206500	0.487553	0.820680	0.775362	0.794063	0.757522

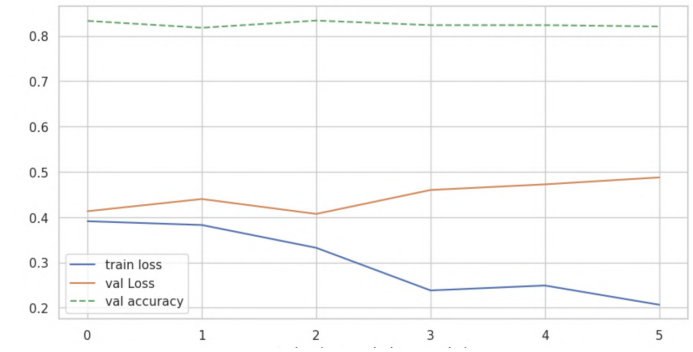

FIGURE A.1 – Détails et visualisation des performances d'entraînement : **BERT-base batch size 32**

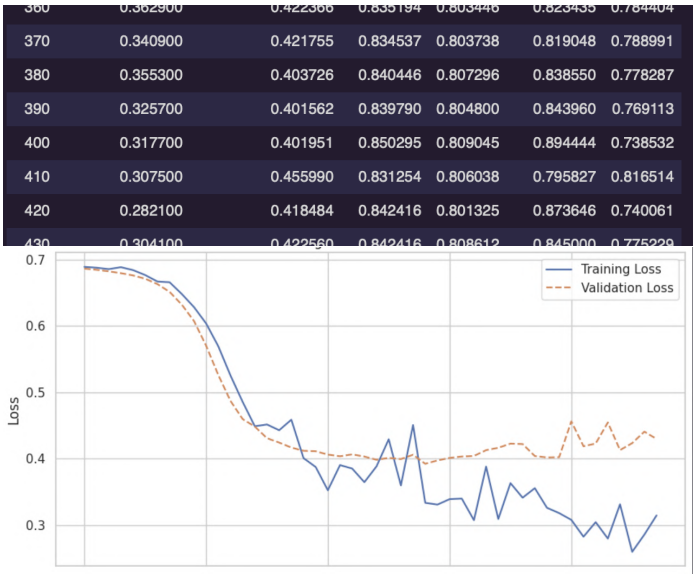

FIGURE A.2 – Détails et visualisation des performances d'entraînement : **BERTweet-base batch size 32**

**Pour cet entraînement du modèle BERTweet avec un batch size de 32, l'affichage des métriques a été configuré différemment :*

```
    num_train_epochs=6,
    learning_rate=1e-5,
    ...
    logging_steps=10,
    evaluation_strategy="steps",
```

FIGURE A.3 – Détails et visualisation des performances d'entraînement : **BERT-base batch size 8**

FIGURE A.4 – Détails et visualisation des performances d'entraînement : **BERTweet-base batch size 8**

A.2 Détails des résultats avec MLM

[1038/1038 01:38, Epoch 6/6]

Epoch	Training Loss	Validation Loss	Accuracy	F1	Precision	Recall
1	0.486700	0.398630	0.819234	0.781086	0.772964	0.789381
2	0.279400	0.393066	0.841649	0.793591	0.848790	0.745133
3	0.323500	0.401758	0.843095	0.797386	0.843874	0.755752
4	0.265400	0.458432	0.838033	0.798561	0.811700	0.785841
5	0.203700	0.489100	0.830803	0.792920	0.792920	0.792920
6	0.166300	0.489846	0.835141	0.794595	0.809174	0.780531

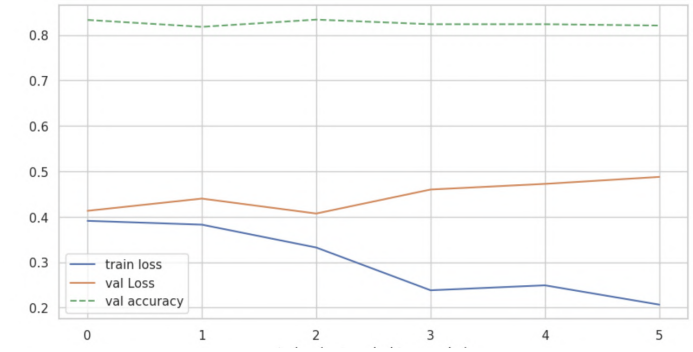

FIGURE A.5 – Détails et visualisation des performances d'entraînement : **BERT-base avec MLM batch size 32**

[3460/3460 06:05, Epoch 5/5]

Epoch	Training Loss	Validation Loss	Accuracy	F1	Precision	Recall
1	0.389000	0.388619	0.848879	0.810517	0.830855	0.791150
2	0.526000	0.436268	0.849602	0.809174	0.840000	0.780531
3	0.265400	0.618286	0.845987	0.802595	0.842412	0.766372
4	0.442100	0.748574	0.827910	0.790861	0.785340	0.796460
5	0.191700	0.811537	0.828633	0.790821	0.788732	0.792920

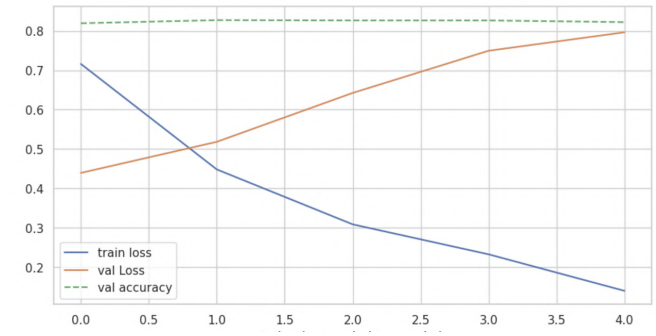

FIGURE A.6 – Détails et visualisation des performances d'entraînement : **BERT-base avec MLM batch size 8**