Note méthodologique : preuve de concept

Dataset retenu

Le dataset choisi pour ce projet provient de Stackoverflow, une plateforme en ligne qui facilite le partage de connaissances et la résolution de problèmes dans le domaine de la programmation informatique. Ce dataset comprend des milliers de questions, réponses et commentaires sur divers sujets liés à la programmation et au développement informatique.

On peut récupérer les données directement depuis stackexchange en utilisant des requêtes SQL. Pour construire un modèle de qualité, j'ai effectué un filtrage initial, sélectionnant uniquement les questions répondant à certains critères :

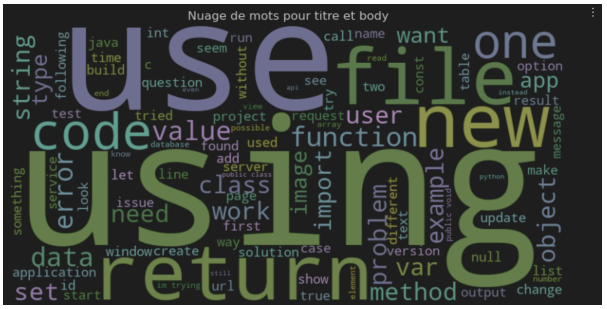
- Possédant au moins 5 tags, puisque notre objectif principal est de prédire des tags.
- Avoir un nombre de vues supérieur à 20.
- Comporter au moins 1 réponse.

Ainsi, cela favorise la sélection des questions les plus populaires en priorité.

Pour cette preuve de concept, nous nous concentrons sur un sous-ensemble des données contenant les 100 tags les plus populaires, afin de simplifier la tâche de classification.

Pour pouvoir être analyser, le dataset est nettoyé via différentes étapes dont la lemmatisation, tokenisation, suppression des stopwords et outliers, vectorisation.

Voici un nuage de mots représentant les termes les plus fréquents dans les documents.



Le dataset nettoyé contient plus de 30000 lignes, il est sauvegardé dans un fichier csv, avec différentes colonnes:

- **title_lemmatized**: titre de la question
- **body_lemmatized**: corps de la question
- **tags_transformed**: catégories / tags associés à la question
- **creationdate**: date de création de la question

Les concepts de l’algorithme récent

XLNet est un modèle de langage pré-entraîné basé sur l'architecture des transformers, développé par Google. Il est conçu pour capturer les dépendances à long terme dans les séquences de texte et est capable de générer des représentations de mots et de phrases de haute qualité.

Il utilise les principes suivants :

Architecture Transformer : XLNet est basé sur l'architecture Transformer, qui utilise des mécanismes d'attention pour capturer les relations entre les mots dans une séquence. Contrairement aux modèles de langage précédents, qui sont entraînés de manière unidirectionnelle ou bidirectionnelle, XLNet utilise une approche permutationnelle, où les mots dans la séquence sont permutés aléatoirement avant d'être alimentés dans le modèle.

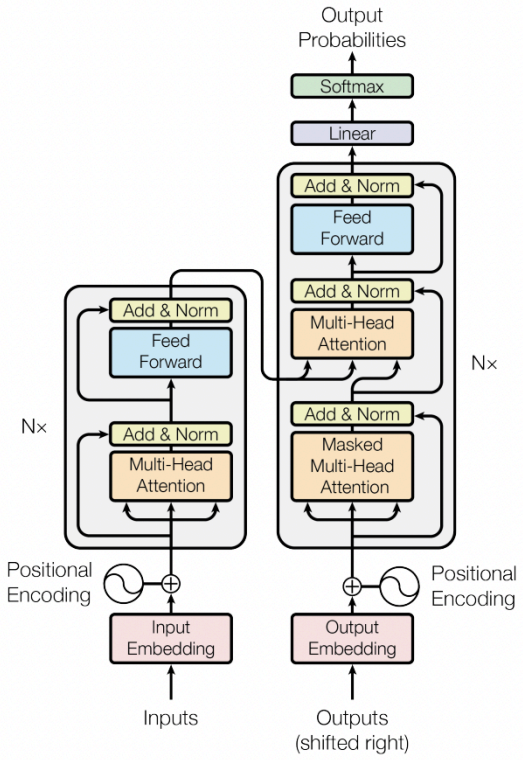


Figure 1: The Transformer - model architecture.

Permutation Language Model (PLM) : Cette approche permet à XLNet d'apprendre des relations entre tous les mots dans une phrase, indépendamment de leur ordre d'apparition. Cela aide le modèle à capturer des dépendances à long terme et à mieux généraliser sur des tâches de compréhension du langage naturel.

Mécanisme d'attention : XLNet utilise des mécanismes d'attention multi-tête pour calculer les pondérations d'attention entre chaque paire de mots dans une séquence. Cela permet au modèle de prendre en compte les interactions complexes entre les mots et de pondérer l'importance des différentes parties de la séquence lors de la génération de représentations.

3.2 Attention

An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum

3

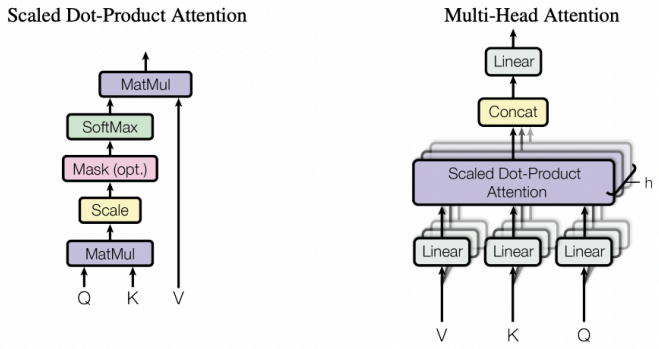


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

Entraînement préalable massif : Avant d'être utilisé pour des tâches spécifiques, XLNet est pré-entraîné sur de vastes corpus de texte non étiquetés. Pendant l'entraînement, le modèle apprend à prédire les mots manquants dans une séquence en se basant sur le contexte environnant, ce qui lui permet de capturer les motifs et les structures du langage naturel.

En résumé, XLNet est un modèle de langage pré-entraîné basé sur l'architecture Transformer, qui utilise une approche permutationnelle pour capturer les dépendances à long terme dans les séquences de texte. En utilisant des mécanismes d'attention et un entraînement préalable massif, XLNet est capable de générer des représentations de texte de haute qualité, adaptées à une variété de tâches de traitement du langage naturel.

La modélisation

Méthodologie de modélisation

Prétraitement des données : Avant de construire les modèles, les données sont prétraitées pour les rendre adaptées à l'apprentissage automatique. Cela inclut des étapes telles que la lemmatisation, la tokenisation, la suppression des stopwords et la vectorisation des données textuelles.

Choix des modèles : Pour ce projet, deux modèles sont utilisés : XLNet et Multinomial Naive Bayes (la baseline). XLNet est un modèle de langage pré-entraîné basé sur les transformers, tandis que Multinomial Naive Bayes est un modèle classique de classification multilabel.

Entraînement des modèles : Les modèles sont entraînés sur les données prétraitées à l'aide de techniques d'apprentissage supervisé. Pendant l'entraînement, les paramètres des modèles sont ajustés pour minimiser une fonction de perte, généralement la perte de log-vraisemblance pour la classification multilabel.

Évaluation des modèles : Une fois les modèles entraînés, ils sont évalués sur un ensemble de test pour estimer leurs performances. Différentes métriques sont utilisées pour évaluer la qualité des prédictions, telles que la précision, le rappel, le score F1.

Métrique d'évaluation retenue

Précision (Precision) : La précision mesure la proportion des éléments positifs (prédits comme positifs) qui sont réellement positifs. Une précision plus élevée indique moins de faux positifs, ce qui est souhaitable dans de nombreux cas.

Rappel (Recall) : Le rappel mesure la proportion des vrais positifs qui sont correctement identifiés parmi tous les éléments réellement positifs. Un rappel plus élevé indique moins de faux négatifs, ce qui est important pour ne pas manquer de vrais positifs.

F1-Score : Le F1-Score est la moyenne harmonique (pour les probabilité) de la précision et du rappel. Il donne une mesure équilibrée entre la précision et le rappel. Un F1-Score élevé indique à la fois une bonne précision et un bon rappel.

Temps d'entraînement : Le temps d'entraînement mesure la durée nécessaire pour entraîner chaque modèle. Il est important de prendre en compte le temps d'entraînement lors de la comparaison des performances des modèles, surtout pour des ensembles de données plus volumineux.

En utilisant ces métriques, on obtient une vue complète des performances des modèles pour choisir le plus adapté à notre besoin. La création d'une interface graphique permettra de comparer les prédictions des deux modèles.

Démarche d'optimisation

La démarche d'optimisation consiste à rechercher les meilleures combinaisons de paramètres et de techniques pour améliorer les performances des modèles.

Les hyper paramètres suivant ont été choisis:

- **learning_rate** : taux d'apprentissage de **2e-5** pour l'optimiseur AdamW utilisé lors de l'initialisation de self.optimizer.
- **epochs** : nombre de fois où l'ensemble des données d'entraînement est passé à travers le modèle complet pendant la phase d'entraînement, le nombre **6** a été choisi.
- **batch_size** : C'est la taille du batch utilisée lors de l'entraînement dans la méthode fit. Le batch a une taille de 4.
- **threshold** : Seuil de probabilité à 0.3 dans l'initialisation de l'objet XLNetPipeline.
-

```
flo *
class XLNetPipeline:
    flo
    def __init__(self, model):
        self.model = model
        self.threshold = 0.3
        self.optimizer = torch.optim.AdamW(model.parameters(), lr=2e-5)
        self.loss_fn = torch.nn.BCEWithLogitsLoss()

    flo
    def fit(self, X, y, epochs=6, batch_size=4):
        if torch.cuda.device_count() > 1:
            print("Utilisation de", torch.cuda.device_count(), "GPUs pour l'entraînement.")
            self.model = DataParallel(self.model)
        for epoch in range(epochs):
            epoch_loss = 0.0
```

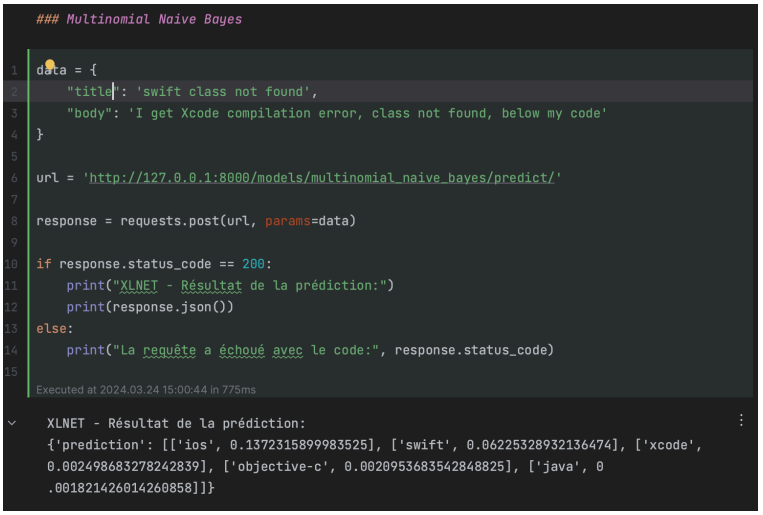
L'entraînement du transformer étant très long, les hyperparamètres ont été modifiés par itération manuellement, en accord avec le nombre de données présentes dans le dataset.

Un processus automatique requiert de plus grosses ressources pour le calcul, et aussi causerait un temps d'entraînement beaucoup plus long (plusieurs jours).

Une synthèse des résultats

Après avoir testé les 2 modèles, plusieurs comparaisons ont été effectuées, tout d'abord manuellement, avec plusieurs dizaines de phrases.

Multinomial Naive Bayes



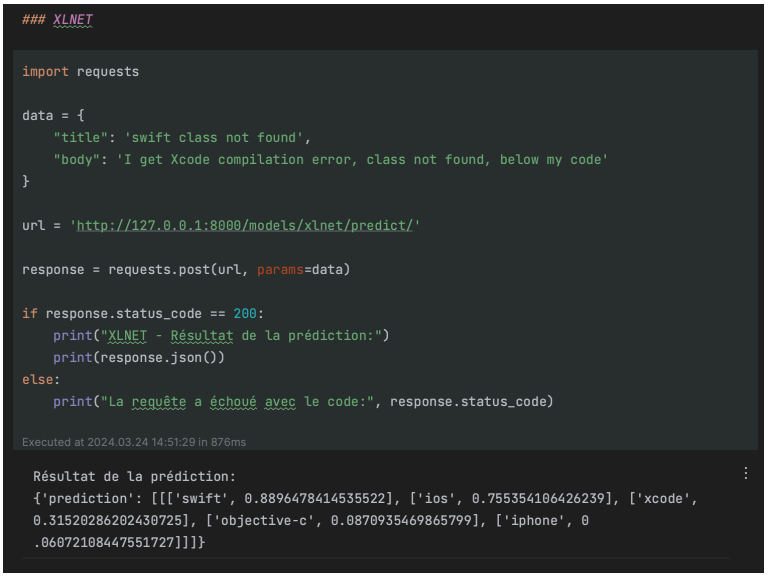
```
### Multinomial Naive Bayes

data = {
    "title": 'swift class not found',
    "body": 'I get Xcode compilation error, class not found, below my code'
}

url = 'http://127.0.0.1:8000/models/multinomial_naive_bayes/predict/'

response = requests.post(url, params=data)

if response.status_code == 200:
    print("XLNET - Résultat de la prédiction:")
    print(response.json())
else:
    print("La requête a échoué avec le code:", response.status_code)

Executed at 2024.03.24 15:00:44 in 775ms

XLNET - Résultat de la prédiction:
{'prediction': [['ios', 0.1372315899983525], ['swift', 0.06225328932136474], ['xcode',
0.002498683278242839], ['objective-c', 0.0020953683542848825], ['java', 0
.0018214260142608581]]}
```

On peut voir dans ce premier exemple, qu'on obtient les probabilités suivantes:

ios: 0.13, swift: 0.06, xcode: 0.002, objective-c: 0.002, java: 0.001

Les catégories prédites par l'algorithme ont donc du sens, car le titre et le body décrivent les technologies liées à l'environnement de développement iOS. Par contre, les probabilités restent avec des valeurs très basses. Dans la prochaine section, on notera que l'algorithme XLNET produit des catégories similaires mais avec une confiance bien meilleure.

XLNET



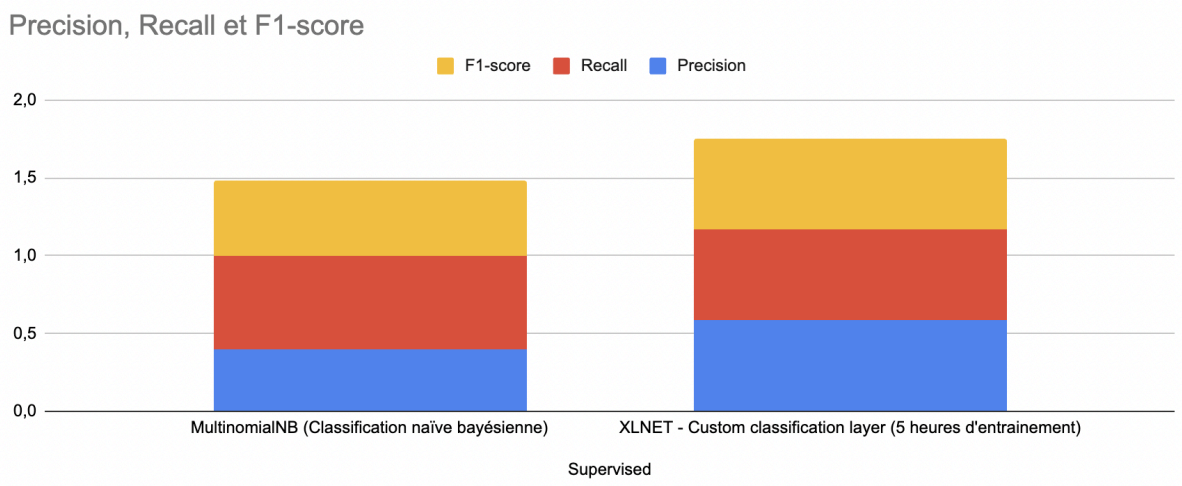
```
### XLNET

import requests

data = {
    "title": 'swift class not found',
    "body": 'I get Xcode compilation error, class not found, below my code'
}

url = 'http://127.0.0.1:8000/models/xlnet/predict/'

response = requests.post(url, params=data)

if response.status_code == 200:
    print("XLNET - Résultat de la prédiction:")
    print(response.json())
else:
    print("La requête a échoué avec le code:", response.status_code)

Executed at 2024.03.24 14:51:29 in 876ms

Résultat de la prédiction:
{'prediction': [[['swift', 0.8896478414535522], ['ios', 0.755354106426239], ['xcode',
0.31520286202430725], ['objective-c', 0.0870935469865799], ['iphone', 0
.06072108447551727]]]}
```

Cette fois avec XLNET, on obtient les probabilités suivantes: **ios: 0.88, swift: 0.75, xcode: 0.31, objective-c: 0.08, java: 0.06**

La première partie sur la comparaison des résultats avec plusieurs dizaines de documents révèle que l'algorithme XLNET prédit de manière plus juste les catégories, mais aussi avec plus de confiance d'un point de vue probabiliste.

Après cette première étape de comparaison, voici les résultats globaux avec les différentes métriques choisies:

Comparaison baseline / XLNET				
	Precision	Recall	F1-score	Temps (minutes)
MultinomialNB (Classification naïve bayésienne)	0,4	0,6	0,48	0,5
XLNET - Custom classification layer (5 heures d'entrainement)	0,59	0,58	0,58	453



En examinant les résultats, on observe que XLNET affiche une amélioration de performance moyenne de 22% par rapport à la référence de base. De plus, ses prédictions se distinguent par leur précision accrue. Alors que MultinomialNB avait tendance à produire des prédictions avec des probabilités souvent inférieures à 0.15, qui étaient étroitement groupées, XLNET présente un intervalle de probabilité plus étendu. Certaines catégories peuvent atteindre des probabilités allant jusqu'à 0.9, et sont nettement différenciées les unes des autres.

Les limites et les améliorations possibles

Malgré les performances prometteuses du transformer XLNET, il existe plusieurs limites et des possibilités d'amélioration afin de gagner en performance.

Limites

Données déséquilibrées : Les déséquilibres dans les ensembles de données peuvent être problématiques lors de l'apprentissage des modèles, car les classes minoritaires risquent d'être sous-représentées. Afin de remédier à ce biais potentiel, le jeu de données a été soigneusement nettoyé et réduit à un nombre restreint de tags pour l'entraînement. Cependant, il est envisageable d'élargir le nombre de tags sur lesquels le modèle est formé, bien que cela nécessite un travail approfondi dans le processus de nettoyage des données.

Interprétabilité : Les modèles complexes comme XLNet ou d'autres transformers peuvent être difficiles à interpréter. Cela signifie qu'il est compliqué de comprendre comment ces modèles prennent leurs décisions. Cette limitation peut rendre leur utilisation difficile dans des domaines où la transparence est importante.

Sensibilité aux hyperparamètres : Les performances du modèle peuvent être sensibles aux hyperparamètres choisis lors de l'entraînement. Trouver les valeurs optimales des hyperparamètres peut être un processus complexe et nécessite souvent des recherches approfondies.

Améliorations possibles

Augmentation des données : Ajouter plus d'exemples, surtout pour les classes moins représentées, aide à mieux former les modèles et à éviter les prédictions biaisées.

Optimisation des hyperparamètres : Une optimisation automatique des hyperparamètres peut être effectuée**.** Par exemple avec la recherche par grille ou avec des modèles probabilistes.

Nombre de données et ressources: Agrandir le dataset permettrait d’obtenir encore plus de données pour entraîner le modèle, cela aurait un impact sur le temps d'entraînement (qui dure déjà plusieurs heures) et sur les nombre de ressources nécessaires pour l’entrainer que ce soit en local ou sur le cloud.

En conclusion, en surmontant les limites telles que les données déséquilibrées et en optimisant les hyperparamètres, il serait possible d'accroître la performance du modèle. Mais cela aurait un grand impact sur le nombre de ressources nécessaires et le temps d'entraînement du modèle.