Évaluation comparative de l'approche Text-Embedding-ADA-002 dans une classification multilabel : un rapport de Proof of Concept

Thomas Mebarki
OpenClassrooms

Sommaire

Introduction :

Dans un monde de plus en plus numérique, la quantité de données textuelles générées et disponibles ne cesse d'augmenter. Ces données, qui proviennent de diverses sources telles que les médias sociaux, articles scientifiques, blogs, etc., regorgent d'informations précieuses. Le traitement du langage naturel (NLP pour “Natural Language Processing”), une branche de l'intelligence artificielle, se concentre sur l'interaction entre les ordinateurs et les humains à travers le langage naturel. L'objectif est de lire, décoder, comprendre et tirer des significations utiles des textes humains. Au cours de la dernière décennie, le NLP a réalisé des avancées significatives, facilitant la gestion et l'analyse de grandes quantités de données textuelles. Cela a un impact direct sur de nombreux domaines tels que la recherche d'informations, la publicité, la recommandation de contenu, entre autres.

Dans ce contexte, Stack Overflow, un site de questions-réponses très populaire parmi les développeurs et les professionnels de la technologie, génère une grande quantité de données textuelles sous forme de questions, réponses et commentaires. Les questions posées sur Stack Overflow sont souvent marquées avec des "tags" qui indiquent les sujets ou les technologies concernés, permettant ainsi un filtrage et une recherche efficaces. Prédire automatiquement ces tags à partir du texte des questions est une tâche importante qui peut améliorer l'efficacité et la pertinence du site.

L'objectif de ce projet est d'évaluer l'efficacité d'une nouvelle méthode de représentation de texte appelée text-embedding-ada-002 [1] (désignée par la suite comme ADA) dans la tâche de classification de texte pour prédire les tags de questions sur Stack Overflow. Cette méthode sera comparée à des méthodes de référence : la méthode classique TF-IDF (Term Frequency-Inverse Document Frequency) combinée avec un classificateur One-Vs-Rest (OVR) et une baseline naïve utilisant DummyClassifier. Il est important de noter que la méthode ADA présente une contrainte en ne pouvant traiter que 8192 tokens à la fois, ce qui a nécessité un filtrage des données en amont.

Pour évaluer et comparer les performances de ces méthodes, nous utiliserons une métrique appelée score Jaccard, qui est particulièrement adaptée aux problèmes de classification multilabels tels que la prédiction de tags. De plus, une analyse qualitative sera effectuée en examinant des exemples spécifiques de texte et en évaluant la qualité des prédictions faites par chaque méthode. Pour le prétraitement des données destinées à être utilisées avec ADA, la librairie tiktoken a été employée pour gérer les contraintes liées aux tokens.

En résumé, ce rapport explore et analyse la performance du modèle text-embedding-ada-002 dans le contexte de la classification de texte, en se concentrant spécifiquement sur la prédiction de tags multilabels pour des questions sur Stack Overflow. Cette analyse nous permettra de comprendre les avantages et les limites de cette nouvelle méthode par rapport aux méthodes existantes.

Méthodologie :

Introduction aux méthodes de classification de texte

Dans le domaine du traitement du langage naturel, la classification de texte est une tâche courante qui consiste à attribuer des catégories prédéfinies (ou tags) à un texte donné. Cette section introduira les méthodes de représentation de texte et les classificateurs utilisés dans ce rapport. Text-Embedding-ADA-002 (ADA) est utilisé comme algorithme de représentation de texte sous forme d'embeddings, tandis que TF-IDF est une technique de transformation de texte, généralement qualifiée de "sac de mots" (bag of words). DummyClassifier est un classificateur utilisé comme base de référence. Nous allons détailler le fonctionnement et l'implémentation de ces représentations de texte ainsi que la manière dont elles sont utilisées en conjonction avec des classificateurs pour effectuer la tâche de classification.

Présentation de Text-Embedding-ADA-002 (ADA)

Présentation du modèle

Text-Embedding-ADA-002, que nous appellerons ADA, est un modèle de plongement de texte développé par OpenAI. Ce modèle est basé sur l'architecture des Transformers, qui utilise un mécanisme d'attention pour comprendre le contexte de chaque mot dans une séquence. Ce mécanisme lui permet de convertir des textes en représentations numériques (embeddings) qui capturent les relations sémantiques entre les concepts dans le texte [2]. Ces embeddings peuvent ensuite être utilisés pour diverses tâches de traitement du langage naturel, telles que la classification de texte. ADA est étroitement lié au modèle GPT-3.5 et exploite des fonctionnalités similaires pour générer des embeddings de texte. Il s'agit d'un progrès significatif par rapport aux techniques traditionnelles comme TF-IDF, en raison de sa capacité à comprendre et représenter le contexte d'un mot dans sa séquence.

Implémentation

L'implémentation de la méthode ADA a été réalisée en utilisant la bibliothèque Python fournie par OpenAI. Les embeddings pour chaque question dans l'ensemble d'entraînement et de test ont été générés. Ces embeddings ont ensuite été utilisés pour entraîner un modèle de classification afin de prédire les tags associés à chaque question.

Introduction à la méthode TF-IDF

Présentation du modèle

TF-IDF (Term Frequency-Inverse Document Frequency) est une technique de pondération souvent utilisée en recherche d'information et en traitement du langage naturel. Elle attribue un poids à chaque terme dans un document, ce qui reflète l'importance de ce terme dans le document et l'ensemble du corpus.

Implémentation

Pour implémenter la méthode TF-IDF, la bibliothèque scikit-learn a été utilisée. Un vectoriseur TF-IDF a été ajusté sur l'ensemble d'entraînement et utilisé pour transformer les questions en représentations vectorielles. Ces représentations ont ensuite servi à entraîner un modèle de classification pour prédire les tags.

Introduction à la méthode de baseline DummyClassifier

Présentation du modèle

DummyClassifier est un classificateur qui fait des prédictions en utilisant des règles simples. Cela est souvent utilisé comme un classificateur de base pour comparer les performances des modèles réels.

Implémentation

Un DummyClassifier a été utilisé en tant que classificateur de référence. La bibliothèque scikit-learn a été utilisée pour mettre en œuvre cette méthode. Cela a permis de comparer les performances des méthodes ADA et TF-IDF par rapport à une baseline simple et dite “naïve”.

Résumé de la méthodologie

En synthèse, la méthodologie s’articule autour de trois approche, à savoir Text-Embedding-ADA-002 (ADA), TF-IDF et DummyClassifier. ADA, issu de OpenAI, se focalise sur les représentations numériques des textes, tandis que TF-IDF évalue l’importance des termes dans les documents. DummyClassifier sert de point de référence grâce à sa simplicité.

Dans le chapitre qui suit, nous nous pencherons sur les étapes cruciales d’organisation des données de Stack Overflow pour garantir l’efficacité de ces méthodes.

Préparation des données :

Description du dataset de Stack Overflow utilisé

Avant d’entrer dans les détails techniques des méthodes de classification de texte utilisées, il est important de comprendre la nature des données avec lesquelles nous travaillons. Le dataset utilisé dans cette étude est constitué de 5 000 questions posées sur la plateforme Stack Overflow. Chaque question est accompagnée de tags qui la catégorisent selon différents sujets ou technologies. Ces tags seront utilisés comme labels pour notre tâche de classification multilabels.

Étapes de prétraitement des données pour TF-IDF

Le prétraitement des données est une étape cruciale pour garantir la performance des algorithmes de classification de texte. Pour la méthode TF-IDF, les étapes suivantes ont été appliquées:

- Tokenization: Cette étape consiste à décomposer le texte en mots individuels (tokens). Cela permet à l’algorithme de traiter les mots séparément et d’identifier la structure du texte.
- Filtrage des Stop Words: Les Stop Words sont des mots communs tels que “le”, “et”, “à” qui n’apportent pas d'information significative au texte. Ces mots sont donc supprimés.
- Mise en minuscule (Lower Casing): Cette étape consiste à convertir tout le texte en lettres minuscules afin de garantir que l’algorithme ne traite pas les mots "Python" et "python" comme différents.
- Lemmatisation: Cela implique la conversion des mots à leur forme de base. Par exemple, “courir”, “couru” seront tous convertis en “courir”. Cela aide à réduire la dimensionnalité des données.

Étapes de prétraitement des données pour ADA

Pour le modèle ADA, une approche différente de prétraitement a été utilisée. Au lieu d'utiliser une tokenization standard et d'autres méthodes de prétraitement, la librairie tiktoken a été utilisée avec l'algorithme cl100k_base [3]. Tiktoken est une librairie spécialisée dans la tokenization adaptée aux modèles de langage d'OpenAI, et cl100k_base est un algorithme qui a été optimisé pour travailler avec ce type de modèle.

Il est également important de noter que, selon les spécifications d'ADA, il ne peut traiter que 8192 tokens à la fois [1]. Cela a nécessité un filtrage des données pour s'assurer qu'aucun individu ne dépasse cette limite de tokens.

Conclusion de la préparation des données

En résumé, ce chapitre a mis en évidence l'importance du prétraitement des données et a présenté les étapes spécifiques utilisées pour préparer les données de Stack Overflow pour la classification de texte. Pour TF-IDF, cela comprenait la tokenization, le filtrage des stop words, la mise en minuscules et la lemmatisation. Pour ADA, nous avons utilisé la librairie tiktoken avec l'algorithme cl100k_base.

Dans le chapitre suivant, nous aborderons en détail comment les méthodes de classification de texte, Text-Embedding-ADA-002, TF-IDF, et DummyClassifier, ont été implémentées et intégrées à ce projet.

Implémentations des Méthodes :

Détails de l'implémentation de ADA pour la classification de texte

Pour implémenter ADA dans ce projet, nous avons d'abord préparé les données de texte comme décrit dans le chapitre précédent. Ensuite, nous avons utilisé l'API ADA pour convertir ces textes en vecteurs. En raison de la limite de 8192 tokens, nous avons dû veiller à ce que chaque texte ne dépasse pas cette limite.

Après avoir obtenu les vecteurs de texte, nous les avons utilisés comme fonctionnalités pour entraîner un modèle de classification OneVsRest avec des hyperparamètres spécifiques (C=10, penalty='l1', solver='liblinear'). Ces paramètres ont été choisis pour garantir la comparabilité avec les autres méthodes.

Détails de l'implémentation de TF-IDF pour la classification de texte

Dans ce projet, nous avons utilisé la bibliothèque scikit-learn pour implémenter TF-IDF. Les paramètres spécifiques max_df=1.0, min_df=1 et stop_words=None ont été utilisés, comme cela avait été optimisé lors d'un précédent projet.

Tout comme pour ADA, nous avons ensuite utilisé ces vecteurs de caractéristiques pour entraîner un modèle OneVsRest avec les mêmes hyperparamètres mentionnés précédemment. Ces derniers ont d'ailleurs été configurés lors de ce projet passé dans le but d'optimiser les performances de TF-IDF. ADA sera donc comparé à un TF-IDF sur ses meilleurs hyperparamètres.

Utilisation de DummyClassifier comme baseline naïve

DummyClassifier est utilisé dans ce projet comme une méthode de base pour comparer les performances des autres méthodes. Il s'agit d'un classificateur qui ne tient pas compte des données et classe selon une règle simple, telle que "prédire la classe la plus fréquente". Cela fournit un point de référence sur la performance que l'on obtiendrait avec une approche naïve.

Conclusion de l'implémentation des méthodes

Pour conclure, dans ce chapitre, nous avons décrit en détail l'implémentation des trois méthodes de classification de texte utilisées dans ce projet: Text-Embedding-ADA-002, TF-IDF, et DummyClassifier. Nous avons expliqué comment les données ont été transformées en vecteurs de caractéristiques et comment les modèles de classification ont été entraînés avec ces caractéristiques.

Dans le chapitre suivant, nous discuterons de la métrique d'évaluation utilisée pour évaluer les performances de ces méthodes, à savoir le score Jaccard. Nous expliquerons pourquoi cette métrique est particulièrement adaptée aux tâches de classification multilabels et comment elle a été appliquée dans ce projet.

Métrique d'évaluation :

Introduction au score Jaccard

Pour évaluer la performance de nos méthodes de classification de texte, il est crucial de choisir une métrique appropriée. Dans ce projet, nous avons choisi d'utiliser le score Jaccard, également connu sous le nom d'indice de Jaccard ou de coefficient de similarité de Jaccard. Le score Jaccard est une mesure statistique de la similarité entre deux ensembles. Il est défini comme la taille de l'intersection divisée par la taille de l'union des deux ensembles d'étiquettes. Mathématiquement, le score Jaccard entre deux ensembles A et B est calculé comme suit:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}.$$

où |A ∩ B| représente le nombre d'éléments communs aux deux ensembles et |A ∪ B| représente le nombre total d'éléments uniques dans les deux ensembles.

Explication de son importance dans les problèmes de classification multilabels

Le choix du score Jaccard comme métrique d'évaluation est particulièrement judicieux pour les problèmes de classification multilabels tels que celui que nous abordons. Dans un tel contexte, chaque instance peut appartenir à plusieurs classes. Les méthodes de classification traditionnelles, telles que la précision, le rappel, et le F1-score, ne sont pas toujours appropriées pour évaluer la performance dans les scénarios multilabels car elles sont principalement conçues pour les classifications binaires ou multiclasse.

Le score Jaccard, en revanche, mesure efficacement à quel point les étiquettes prédites pour une instance sont similaires aux étiquettes réelles. Plus le score Jaccard est proche de 1, plus les étiquettes prédites sont similaires aux étiquettes réelles. Un score Jaccard de 0 indique qu'il n'y a aucune étiquette en commun entre les prédictions et les vraies étiquettes.

Application du score Jaccard dans ce projet

Dans le cadre de ce projet, le score Jaccard a été utilisé pour évaluer les performances de Text-Embedding-ADA-002, TF-IDF, et DummyClassifier. En comparant les scores Jaccard obtenus, nous pouvons avoir une idée claire de la précision de chaque méthode dans la prédiction des tags appropriés pour les questions de Stack Overflow.

Conclusion

En somme, le score Jaccard s'est avéré être une métrique d'évaluation adéquate pour ce projet de classification de texte multilabel. Il a permis d'obtenir une mesure quantitative de la performance de chaque méthode en termes de similitude entre les étiquettes prédites et les étiquettes réelles.

Dans le chapitre suivant, nous aborderons les résultats obtenus en utilisant les méthodes de classification Text-Embedding-ADA-002, TF-IDF, et DummyClassifier. Nous présenterons les résultats quantitatifs et qualitatifs, et nous engagerons une discussion approfondie sur les performances de chaque méthode, en soulignant leurs

Résultats et analyse comparative :

Résultats quantitatifs

Tout d'abord, intéressons-nous aux résultats quantitatifs obtenus par les trois méthodes de classification : ADA, TF-IDF, et DummyClassifier.

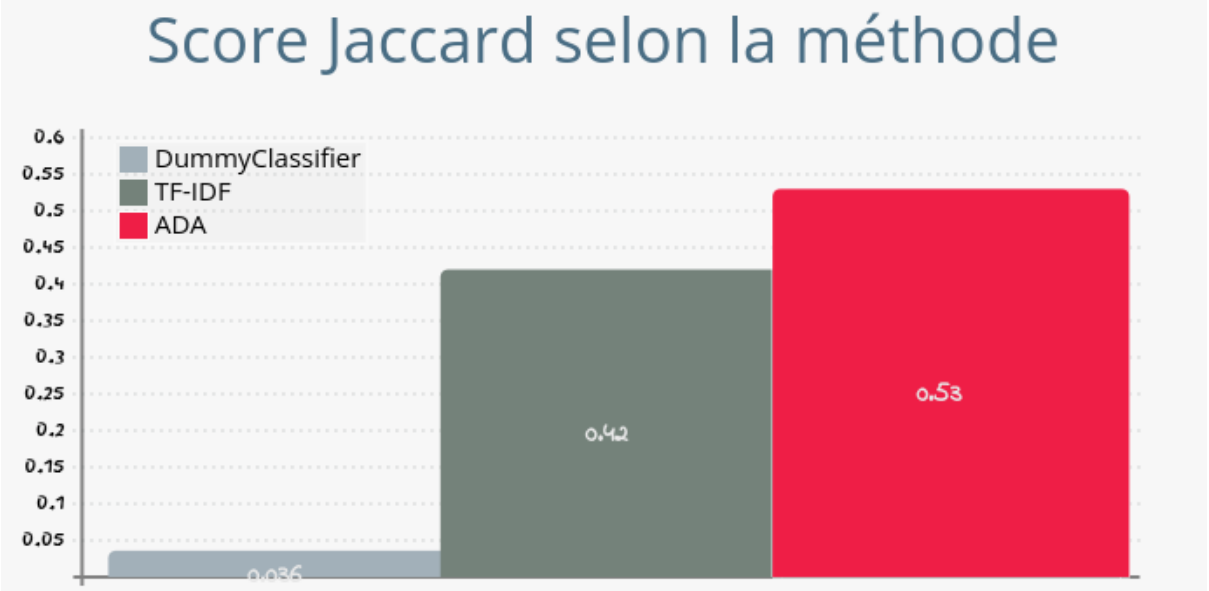

Figure 1: Graphique à barres illustrant la comparaison des scores Jaccard obtenus par les trois méthodes.

DummyClassifier :

Le DummyClassifier a obtenu un score Jaccard de 0.036. Ce résultat était attendu car cette méthode ne tient pas compte des caractéristiques du texte et attribue les labels de manière aléatoire.

TF-IDF :

L'approche TF-IDF, qui est une méthode classique et répandue pour la classification de texte, a obtenu un score Jaccard de 0.42. Cela montre que cette méthode est significativement meilleure que la baseline naïve, en capturant certains aspects informatifs du texte.

ADA (text-embedding-ada-002) :

Enfin, le modèle ADA a obtenu un score Jaccard de 0.53. Cela indique que cette méthode a surpassé les autres en termes de performance, suggérant une meilleure capacité à capturer les relations sémantiques dans le texte et à attribuer les labels de manière plus précise.

Résultats qualitatifs

Exemples de tags prédits par les différentes méthodes :

```
Exemple 479:
Titre de la question  : optimized implementation java.util.map java.util.set
Vrais tags  : java | performance
Tags prédits par dummy: .net | ruby
Tags prédits par classic: java (0.999) | performance (0.847) | c (0.106)
Tags prédits par ada_text: java (0.993) | performance (0.900) | algorithm (0.186) | c# (0.181)
--------------------------------------------------
Exemple 900:
Titre de la question  : layered database application without using orm specify need data display
Vrais tags  : c#
Tags prédits par dummy: jquery
Tags prédits par classic: asp.net (0.653) | sql (0.458) | c# (0.450) | database (0.160)
Tags prédits par ada_text: c# (0.928) | .net (0.576) | database (0.153)
--------------------------------------------------
Exemple 898:
Titre de la question  : angular get object array
Vrais tags  : angular | arrays | json
Tags prédits par dummy: .net | php | sql
Tags prédits par classic: .net (0.267) | c# (0.238) | performance (0.126) | arrays (0.003) | angular (0.001) | jso
n (0.001)
Tags prédits par ada_text: javascript (0.843) | angular (0.364) | arrays (0.137) | json (0.023)
--------------------------------------------------
Exemple 438:
Titre de la question  : how use nszombie xcode
Vrais tags  : iphone | objective-c | xcode
Tags prédits par dummy: javascript | php | python
Tags prédits par classic: xcode (0.986) | ios (0.149) | c++ (0.115) | c (0.112) | iphone (0.037) | objective-c (0.
014)
Tags prédits par ada_text: objective-c (0.283) | xcode (0.266) | macos (0.145) | iphone (0.121)
--------------------------------------------------
Exemple 943:
Titre de la question  : java bytecode optimizer remove useless gotos
Vrais tags  : java
Tags prédits par dummy: ios | linux
Tags prédits par classic: java (0.898)
Tags prédits par ada_text: java (0.946) | c# (0.258) | performance (0.127) | gcc (0.104)
```

Dans cet affichage, on présente les résultats de la prédiction de tags pour des questions, en utilisant différents modèles: "dummy", "classic", et "ada_text". Pour chaque exemple, on montre le titre de la question, les vrais tags associés à cette question, et les tags prédits par chaque modèle. Les tags prédits par le modèle "dummy" sont colorés en vert s'ils correspondent aux vrais tags et en jaune s'ils sont incorrects. Pour les modèles "classic" et "ada_text", les probabilités associées à chaque tag prédit sont également affichées. Les tags réels non prédits (avec une probabilité inférieur à 0.1) sont affichés en rouge.

Analyse qualitative des prédictions

En examinant les résultats, il est manifeste que le modèle "dummy" n'a pas de capacité significative à prédire les tags appropriés, comme en témoignent ses prédictions colorées en jaune, ce qui est attendu pour un modèle de base. Cependant, le focus principal devrait être sur la comparaison entre les modèles "classic" et "ada_text". Le modèle "classic" fait de bonnes prédictions, mais il semble avoir une tendance à prédire certains tags avec des probabilités moindres, même lorsqu'ils sont corrects. Cela pourrait être dû à la nature de la représentation TF-IDF qui accorde de l'importance aux termes en fonction de leur fréquence dans le document mais pas nécessairement de leur pertinence contextuelle. D'autre part, le modèle "ada_text" semble avoir une meilleure performance, prédire des tags pertinents avec des probabilités plus élevées, ce qui indique une plus grande confiance dans ses prédictions. Ceci peut s'expliquer par le fait que "ada_text" est probablement capable de capturer des relations plus complexes et des dépendances contextuelles dans les données, ce qui conduit à des prédictions plus précises et fiables. Cela souligne l'importance d'utiliser des modèles plus avancés lorsqu'on travaille avec des données textuelles complexes.

Conclusion

En résumé, les résultats quantitatifs montrent que le modèle Text-Embedding-ADA-002 (ADA) surpasse les autres méthodes en termes de score Jaccard, indiquant une meilleure précision dans la prédiction des tags pour les questions de Stack Overflow. Le DummyClassifier, utilisé comme une baseline, présente une performance faible comme attendu, tandis que TF-IDF montre des résultats satisfaisants mais inférieurs à ceux d'ADA. L'analyse qualitative révèle que, bien que TF-IDF fasse de bonnes prédictions, ADA semble avoir une plus grande confiance dans ses prédictions et capte des relations plus complexes dans le texte. Cette différence suggère que ADA est plus efficace pour comprendre les nuances contextuelles des données textuelles.

Dans la section suivante, intitulée "Discussion", nous approfondirons notre analyse des résultats, en explorant les points forts et les limites de chaque méthode. Nous discuterons également des raisons pour lesquelles ADA a pu obtenir de meilleures performances et examinerons les implications de ces résultats pour des applications futures et des recherches dans le domaine du Traitement Automatique du Langage Naturel.

Discussion :

Analyse des résultats, point forts et limites de chaque méthode

text-embedding-ada-002 :

Les résultats obtenus par le modèle ADA sont encourageants. L'un des principaux points forts de ce modèle est sa capacité à capter des relations complexes dans le texte, ce qui lui permet de mieux comprendre les nuances contextuelles. Cette propriété est probablement due à la nature de son architecture, qui est basée sur le modèle GPT-3.5. Cependant, il est important de noter que le modèle ADA a des contraintes en termes de nombre de tokens qu'il peut traiter, ce qui a nécessité un prétraitement pour réduire la taille des textes d'entrée.

Lors d'un test de classification d'image réalisé par OpenAI, ADA-002 a obtenu un score de 90.1, tandis qu'un autre modèle également entraîné par OpenAI, nommé Text-Similarity-DaVinci-001, a obtenu un score légèrement supérieur de 92.2 [1]. Cependant, il est important de nuancer ces résultats en considérant les coûts et les contraintes associés à chaque modèle. Le coût de ADA v2 est de 0,00001$ pour 1000 tokens, tandis que DaVinci est à 0,12$ pour 1000 tokens, ce qui est 1200 fois plus cher [4]. De plus, ADA est plus rapide et peut traiter jusqu'à 8192 tokens à la fois, alors que DaVinci est limité à 2049 tokens, soit quatre fois moins. Ces différences sont importantes à prendre en compte, en particulier dans des scénarios d'application en temps réel où les coûts et la rapidité de traitement sont des critères cruciaux.

TF-IDF :

Bien que TF-IDF ait montré des résultats satisfaisants, il n'atteint pas les performances du modèle ADA. Les points forts de TF-IDF incluent sa simplicité et sa facilité de mise en œuvre. Toutefois, en tant que modèle basé sur des bags of words, il ne capte pas efficacement les relations séquentielles dans le texte, ce qui peut expliquer sa performance inférieure par rapport à ADA.

DummyClassifier :

Comme prévu, DummyClassifier a présenté une performance faible. Il est important en tant que baseline pour mettre en contexte les performances des autres méthodes, mais il n'est pas approprié pour une application réelle.

Explication de la performance relative de text-embedding-ada-002

La supériorité d'ADA par rapport à TF-IDF peut s'expliquer par le fait que ADA utilise une approche de plongement de mots qui capte les dépendances contextuelles, contrairement à TF-IDF qui traite les mots indépendamment. De plus, l'architecture sous-jacente de GPT-3.5, sur laquelle ADA est basé, est connue pour sa capacité à comprendre et à générer du texte de manière cohérente, ce qui peut contribuer à sa performance élevée [5].

Implications pour les applications futures

Ces résultats suggèrent que l'utilisation de modèles de plongement de mots tels que ADA pourrait améliorer la performance dans des tâches de classification de texte complexes. Toutefois, il est important de prendre en compte les contraintes de ces modèles,

telles que la limitation du nombre de tokens, lors de leur intégration dans des applications en temps réel, et leur prix qui est certes faible comparé à d'autres mais n'est pas gratuit à contrairement à TF-IDF.

Conclusion

En conclusion, Text-Embedding-ADA-002 apparaît comme une méthode prometteuse pour la classification de texte, en particulier pour des tâches qui exigent une compréhension approfondie du contexte. Sa performance relative, sa rapidité et son coût en font un candidat attrayant. Cependant, il est nécessaire de considérer ses limitations et d'explorer des optimisations pour son intégration efficace dans des systèmes de production.

Dans la prochaine section, nous résumerons les principales conclusions de cette étude et discuterons des perspectives et des recommandations pour des recherches et des applications futures dans le domaine du Traitement Automatique du Langage Naturel.

Conclusion :

Synthèses des résultats

Tout au long de cette étude, nous avons examiné l'efficacité de la méthode Text-Embedding-ADA-002 (ADA) pour la classification multilabel de questions issues de Stack Overflow et avons comparé ses performances avec celles de la méthode TF-IDF et d'un DummyClassifier. Les résultats ont montré que ADA surpasse les autres méthodes en termes de score Jaccard, indiquant qu'elle est plus précise dans la prédiction de tags appropriés pour les questions.

ADA s'est avéré être particulièrement efficace pour comprendre le contexte des questions, ce qui a probablement contribué à son succès dans cette tâche. Bien que TF-IDF ait également performé de manière respectable, il ne parvient pas à atteindre le niveau de précision de ADA, probablement en raison de l'incapacité de TF-IDF à capturer des relations contextuelles complexes. DummyClassifier, étant une baseline naïve, a eu des performances considérablement plus basses, comme prévu.

Vers de nouvelles frontières: Recommandations et Perspectives

Bien que les résultats soient prometteurs, il existe des pistes d'amélioration et des questions qui méritent d'être explorées davantage. Tout d'abord, il serait intéressant d'examiner comment ADA se comporte avec des ensembles de données de tailles différentes et dans différents domaines, pour évaluer sa polyvalence. De plus, des recherches supplémentaires sur l'ajustement des hyperparamètres spécifiques à ADA pourraient conduire à de meilleures performances.

Un autre point d'intérêt est l'intégration des développements récents dans le domaine. Notamment, la technologie "lightspeedEmbeddings" [6] permet d'exploiter le multithreading pour obtenir des embeddings vectoriels jusqu'à 10 fois plus rapidement, ce qui peut être particulièrement utile dans des applications de production où la vitesse est cruciale. Étant donné que "lightspeedEmbeddings" est spécifiquement conçu pour le modèle text-ada-002 d'OpenAI, il se présente comme un complément naturel qui peut améliorer l'efficacité d'ADA.

Il serait également pertinent d'explorer comment ADA peut être intégré efficacement dans des systèmes de production, compte tenu de ses limitations en termes de nombre de tokens et de coûts potentiels. Enfin, il serait intéressant d'évaluer ADA par rapport à d'autres techniques de pointe en NLP pour avoir un aperçu plus complet de sa position relative dans le domaine.

Clôture et Réflexions finales

En somme, cette étude a révélé le potentiel de Text-Embedding-ADA-002 en tant qu'outil de classification de texte multilabels. Ses performances supérieures, notamment dans la compréhension du contexte, en font une option attrayante pour les applications de Traitement Automatique du Langage Naturel. Cependant, comme avec toute technologie, il est impératif de peser soigneusement les avantages et les inconvénients, et de procéder à des investigations supplémentaires pour maximiser son efficacité. Avec des

recherches continues, l'intégration de technologies complémentaires telles que "lightspeedEmbeddings", et des développements futurs, ADA et des méthodes similaires pourraient devenir des éléments incontournables dans le domaine du NLP.

Annexes :

[1] R. Greene, T. Sanders, L. Weng, A. Neelakantan. “New and improved embedding model.”, 15 December 2022, https://openai.com/blog/new-and-improved-embedding-model

Présentation de ada-text-embedding et comparaison avec d’autres modèles de OpenAI

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin “Attention Is All You Need.” *Arxiv*, https://arxiv.org/pdf/1706.03762.pdf

Article scientifique d’introduction aux modèles Transformers.

[3] OpenAI. “Librairie tiktoken.” *GitHub*, https://github.com/openai/tiktoken.

Librairie Tiktoken utilisée pour pré-traîter les données

[4] OpenAI. “Pricing.” *OpenAI*, https://openai.com/pricing.

Annonce les prix des différents modèles disponibles

[5] Wiggers, Kyle. “OpenAI releases GPT-3.5.” *TechCrunch*, 2 December 2022, https://techcrunch.com/2022/12/01/while-anticipation-builds-for-gpt-4-openai-quietly-releases-gpt-3-5/.

Annonce et présente GPT 3.5 sur lequelle repose ada

[6] AndrewGCodes. “Lightspeed Embedding Github.” *github.com*, 06 2023, https://github.com/andrewgcodes/lightspeedEmbeddings/blob/main/README.md.

Présente une nouvelle technologie permettant d’accélérer le processus d’embedding