DIAW ANNA

RAPPORT DE TD NLP

PARTIE I : TEXT CLASSIFICATION: PRÉDIRE SI LA VIDÉO EST UNE CHRONIQUE COMIQUE

Nature du problème : Nous chercheons à prédire si une vidéo est une chronique comique en fonction de son nom (video_name). Cela ressemble à une tâche de classification binaire où vous avez deux classes possibles : chronique comique (is_comic = 1) ou non (is_comic = 0).

Caractéristiques des données : Les données comprennent principalement des noms de vidéos (video_name) et des étiquettes indiquant si la vidéo est une chronique comique (is_comic). Nous avons également des noms de comiques (comic_name) lorsque la vidéo est une chronique comique. Nous disposons d'une représentation binaire de chaque mot du nom de la vidéo indiquant s'il s'agit du nom d'une personne (is_name). Par exemple, "[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]" signifie que les deux derniers mots du nom de la vidéo sont des noms de personnes.

A priori sur les features et les modèles :

Les classes ***CountVectorizer***, ***HashingVectorizer***, et ***TfidfVectorizer*** de **sklearn.feature_extraction.text** semblent toutes adaptées à l'expérimentation pour prédire si une vidéo est une chronique comique. Dans cette section je vais expliquer plus en détail pourquoi je les trouve pertinente pour mon objectif d'étude et expliquer les apports individuels après expérimentation.

****CountVectorizer**** : convertit une collection de documents textuels en une matrice de comptage de tokens. Chaque colonne de la matrice représente un mot unique et chaque cellule contient le nombre d'occurrences de ce mot dans un document. L'utilisation de *CountVectorizer* est appropriée car elle permet de capturer la fréquence des mots dans les documents. Les mots qui apparaissent fréquemment dans les chroniques comiques peuvent être des indicateurs importants.

*****HashingVectorizer**** : est une alternative à CountVectorizer qui convertit des documents en une matrice de fréquence de hachage (hashing trick). Cette classe est adaptée lorsqu'on a un grand nombre de features potentielles (mots) et que l'on réduire la dimensionnalité de la matrice résultante. Elle peut être utile si on travaille avec un grand corpus de textes.

******TfidfVectorizer**** : Cette classe combine les fonctionnalités de CountVectorizer et TfidfTransformer. Elle convertit une collection de documents en une matrice de features TF-IDF. TfidfVectorizer est adapté car il capture à la fois la fréquence des mots et leur importance relative, ce qui peut être bénéfique pour la classification de texte.

Nous cherchons à résoudre la tâche de classification binaire, et il est logique de commencer par des modèles pré-réseaux de neurones, en particulier avec les modèles ***LogisticRegression*** et ***RandomForest*** .

******Logistic Regression**** : La régression logistique est un modèle de classification linéaire simple qui fonctionne bien pour les tâches de classification binaire. Il fournit une ligne de base solide pour la

classification de texte et est facile à interpréter. Il est adapté pour comprendre l'impact des caractéristiques individuelles sur la prédiction, ce qui est important si nous souhaitons des résultats interprétables.

Random Forest* : Les forêts aléatoires sont un ensemble d'arbres de décision qui fonctionnent bien pour la classification binaire, en particulier lorsque le texte a des caractéristiques complexes. Elles sont robustes et peuvent gérer des ensembles de données plus importants avec des caractéristiques non linéaires. Elles ont tendance à bien généraliser même avec des ensembles de données bruyants.

Résultats obtenus :

```
Annas-MacBook-Pro:NLP annadiaw$ python3 src/main.py evaluate --task=is_comic_video --input_file=src/data/raw/train.csv --model_dump=src/model
Evaluating model: LogisticRegression, vectorizer: count
fitted
Model: LogisticRegression
Vectorizer Type: count
Accuracy: 0.96
Precision: 0.92
Recall: 0.79
F1 Score: 0.85
Confusion Matrix:
 [[169   2]
 [  6  23]]
Evaluating model: LogisticRegression, vectorizer: hashing_vectorizer
fitted
Model: LogisticRegression
Vectorizer Type: hashing_vectorizer
Accuracy: 0.90
Precision: 0.80
Recall: 0.41
F1 Score: 0.55
Confusion Matrix:
 [[168   3]
 [ 17  12]]
Evaluating model: LogisticRegression, vectorizer: tfidf_vectorizer
fitted
Model: LogisticRegression
Vectorizer Type: tfidf_vectorizer
Accuracy: 0.86
Precision: 0.60
Recall: 0.10
F1 Score: 0.18
Confusion Matrix:
 [[169   2]
 [ 26   3]]
Evaluating model: RandomForest, vectorizer: count
fitted
Model: RandomForest
Vectorizer Type: count
Accuracy: 0.95
Precision: 0.91
Recall: 0.72
F1 Score: 0.81
Confusion Matrix:
 [[169   2]
 [  8  21]]
Evaluating model: RandomForest, vectorizer: hashing_vectorizer
fitted
Model: RandomForest
Vectorizer Type: hashing_vectorizer
Accuracy: 0.95
Precision: 0.88
Recall: 0.76
F1 Score: 0.81
Confusion Matrix:
 [[168   3]
 [  7  22]]
Evaluating model: RandomForest, vectorizer: tfidf_vectorizer
fitted
Model: RandomForest
Vectorizer Type: tfidf_vectorizer
Accuracy: 0.93
Precision: 0.89
Recall: 0.59
F1 Score: 0.71
Confusion Matrix:
 [[169   2]
 [ 12  17]]
```

D'après les

résultats obtenus pour les modèles "LogisticRegression" et "RandomForest" avec différentes vectorization methods (count, hashing_vectorizer, tfidf_vectorizer), nous pouvons en déduire que :

Logistic Regression* :

Le modèle **LogisticRegression** avec la vectorization de type "**count**" a obtenu les **meilleures performances** parmi les trois vectorization methods. Il a une précision de 0.92, un rappel de 0.79 et un score F1 de 0.85. Il est également très précis, avec une précision de 0.96.

Cela suggère que la vectorization "count" est bien adaptée à ce problème de classification de texte.

Le modèle LogisticRegression avec la vectorization de type "hashing_vectorizer" a montré des performances légèrement inférieures, avec un rappel de seulement 0.41. Cela signifie qu'il a tendance à manquer de vidéos comiques potentielles.

Le modèle LogisticRegression avec la vectorization de type "tfidf_vectorizer" a obtenu les performances les plus faibles, avec un rappel de seulement 0.10. Cela signifie qu'il manque la plupart des vidéos comiques.

Random Forest* :

Le modèle RandomForest a obtenu de bonnes performances avec toutes les vectorization methods. Cependant, il a des performances légèrement inférieures par rapport au modèle LogisticRegression avec la vectorization “count."

Le modèle RandomForest avec la vectorization de type "count" a une précision de 0.91, un rappel de 0.72, et un score F1 de 0.81. Il est précis mais a un rappel légèrement inférieur par rapport à LogisticRegression “count."

Le modèle RandomForest avec la vectorization de type "hashing_vectorizer" a des performances similaires au modèle “count."

Le modèle RandomForest avec la vectorization de type "tfidf_vectorizer" a des performances légèrement inférieures.

Conclusions :

L'une des premières étapes de notre démarche a consisté à explorer les caractéristiques potentiellement utiles pour la classification des vidéos humoristiques. Nous avons examiné trois types de vectorisation des textes : "Count Vectorizer", "Hashing Vectorizer" et "TF-IDF Vectorizer". L'utilisation de ces différentes approches nous a permis de découvrir des nuances intéressantes dans la représentation des données textuelles.

L'analyse de ces features a révélé que la "Count Vectorization" a généré des représentations riches des vidéos humoristiques. En utilisant cette technique, notre modèle "LogisticRegression" a atteint une précision de 0,92 et un rappel de 0,79. Cela indique que la représentation en nombre d'occurrences de mots a bien fonctionné pour identifier les vidéos comiques. Cependant, il est important de noter que la "TF-IDF Vectorization" a eu du mal à capturer les informations discriminantes, avec une précision de seulement 0,60.

Nous avons effectué une comparaison détaillée des performances de deux modèles : "LogisticRegression" et "RandomForest". La "LogisticRegression" a démontré une précision globalement supérieure, en particulier lors de l'utilisation de "Count Vectorization". Cependant, "RandomForest" a montré une capacité notable à augmenter le rappel, avec un équilibre entre précision et rappel. Les performances de ces modèles ont été illustrées dans le tableau des matrices de confusion et dans les métriques telles que le score F1.

Nous avons également envisagé la possibilité d'optimiser les hyperparamètres des modèles pour améliorer davantage leurs performances. Une telle optimisation pourrait permettre de trouver le meilleur compromis entre la précision et le rappel, en fonction de nos besoins spécifiques.

L'analyse des erreurs de classification est un aspect essentiel de notre réflexion. Elle nous a permis d'identifier les domaines où nos modèles ont eu du mal. Par exemple, nous avons constaté que certaines vidéos humoristiques ont été mal classées, ce qui souligne la complexité de la tâche de classification. Ces erreurs peuvent être liées à la présence de sarcasme ou d'humour subtil dans les vidéos.

Nous avons examiné la possibilité d'utiliser des techniques d'ensemble learning pour améliorer les performances. Les modèles ensemblistes pourraient potentiellement combiner les forces des modèles individuels pour une meilleure précision tout en maintenant un bon rappel.

L'interprétabilité des modèles est un facteur important à considérer. La "LogisticRegression" offre une interprétabilité plus élevée grâce à ses coefficients, ce qui peut être un avantage si nous devons expliquer les prédictions à un public non technique.

Nous envisageons plusieurs directions futures pour améliorer notre modèle. Tout d'abord, l'exploration de nouvelles features pourrait inclure des informations telles que la longueur du texte, la fréquence des mots clés spécifiques et des caractéristiques plus avancées telles que l'analyse de sentiment. Nous pourrions également considérer l'utilisation de techniques de prétraitement plus avancées, telles que l'utilisation de word embeddings pour capturer des informations sémantiques.

En conclusion, notre démarche pour résoudre la tâche de classification des vidéos humoristiques s'est avérée fructueuse. Nous avons exploré diverses features, modèles et méthodes d'optimisation pour atteindre des performances prometteuses. Cependant, il existe encore des opportunités d'amélioration, notamment dans l'optimisation des hyperparamètres et l'exploration de nouvelles features. Nos conclusions initiales nous incitent à envisager une approche ensembliste pour combiner les forces des différents modèles. Nous prévoyons également de poursuivre notre exploration des modèles pré-réseaux de neurones pour élargir notre boîte à outils de modélisation.

Essaies échoués à explorer :

Nous avons essayer de tester le modèle MultinomialB mais il s'est avéré non fructueux étant donné qu'il y'avait une erreur que nous n'avons pas pu débogguer par manque de temps. C'est une piste à explorer..

```
Annas-MacBook-Pro:NLP annadiaw$ python3 src/main.py train --task=is_comic_video --input_file=src/data/raw/tra
in.csv --model_dump=src/model
Training model: LogisticRegression, vectorizer: count
Training model: LogisticRegression, vectorizer: hashing_vectorizer
Training model: LogisticRegression, vectorizer: tfidf_vectorizer
Training model: MultinomialNB, vectorizer: count
Training model: MultinomialNB, vectorizer: hashing_vectorizer
Traceback (most recent call last):
  File "/Users/annadiaw/Desktop/NLP/src/main.py", line 111, in <module>
    cli()
  File "/opt/homebrew/lib/python3.11/site-packages/click/core.py", line 1157, in __call__
    return self.main(*args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/opt/homebrew/lib/python3.11/site-packages/click/core.py", line 1078, in main
    rv = self.invoke(ctx)
         ^^^^^^^^^^^^^^^^
  File "/opt/homebrew/lib/python3.11/site-packages/click/core.py", line 1688, in invoke
    return _process_result(sub_ctx.command.invoke(sub_ctx))
                           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/opt/homebrew/lib/python3.11/site-packages/click/core.py", line 1434, in invoke
    return ctx.invoke(self.callback, **ctx.params)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/opt/homebrew/lib/python3.11/site-packages/click/core.py", line 783, in invoke
    return __callback(*args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/Users/annadiaw/Desktop/NLP/src/main.py", line 30, in train
    model.fit(df)
  File "/Users/annadiaw/Desktop/NLP/src/model/dumb_model.py", line 25, in fit
    self.model.fit(X, y)
  File "/opt/homebrew/lib/python3.11/site-packages/sklearn/base.py", line 1152, in wrapper
    return fit_method(estimator, *args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/opt/homebrew/lib/python3.11/site-packages/sklearn/naive_bayes.py", line 772, in fit
    self._count(X, Y)
  File "/opt/homebrew/lib/python3.11/site-packages/sklearn/naive_bayes.py", line 894, in _count
    check_non_negative(X, "MultinomialNB (input X)")
  File "/opt/homebrew/lib/python3.11/site-packages/sklearn/utils/validation.py", line 1489, in check_non_nega
tive
    raise ValueError("Negative values in data passed to %s" % whom)
ValueError: Negative values in data passed to MultinomialNB (input X)
```