ROQUE louis Kinsley 01/11/2023

ALIZAY Julian



**TD**

Table des matières

[**Partie 1: Text classification: prédire si la vidéo est une chronique comique** 3](#_Toc150097140)

[**Partie 2 : Reconnaître les noms de personnage dans le texte** 4](#_Toc150097141)

[a) Choix du Modèle 4](#_Toc150097144)

b) Pour conclure ……………………………………………………………………6

[**Partie 3 : Assembler les modèles** 7](#_Toc150097150)

1. Construction pour l’assemblement des modèles…………………………………8
2. Adaptation du test et de train ………………………………………………………9
3. Conclusion…………………………………………………………………………10

[**Partie 1: Text classification: prédire si la vidéo est une chronique comique**](https://github.com/afouchet/nlp_esgi/tree/main#partie-1-text-classification-prédire-si-la-vidéo-est-une-chronique-comique)

Le premier parti consiste à prédire grâce à nos données si la vidéo est une chronique humoristique grâce au titre de la vidéo.

Pour cela nous avons une base qu’on va entrainer qui contient 4 champs :

* video\_name: le nom de la video
* is\_comic: est-ce une chronique humoristique
* is\_name: une liste de taille (nombre de mots dans video\_name) valant 1 si le mot est le nom d'une personne, 0 sinon
* comic: le nom du comique si c'est une chronique humoristique

Pour cela nous avons fait un feature simple qui permet de retourner le fait si des données sont présentes dans les colonnes is\_name,is\_comic et comic.

Puis nous avons ensuite optimisé les features faites dans "make\_features ".

Nous avons d’abord utilisé des features pour prétraiter les titres de vidéos en supprimant les caractères indésirables tels que les symboles. Pour cela nous avons créé une fonction (preprocess\_text) qui permet de traite un texte d'entrée pour une analyse ultérieure. La fonction permet de convertit le texte en minuscules, Supprime les caractères accentués, conserve que les lettres de l'alphabet et les espaces, éliminant tous les autres caractères, supprime les mots vides.

Nous avons donc converti les mots en leur forme de base et en supprimant les mots courants sans signification significative. En résumé nous avons préparé des données textuelles avant de les utiliser pour entraîner des modèles.

Nous avons ensuite utilisé la fonction make features qui permets de génère des caractéristiques à partir du nom de la vidéo dans le dataframe. Elle applique la fonction de prétraitement créer précédemment à la colonne "video\_name". Elle renvoie le texte qui sera prétraité et la sortie en fonction de la tâche. Cela permet de préparer les données pour une étape de prédiction ou d'entraînement.

Nous avons ensuite utilisé plusieurs modèles pour avoir plusieurs résultats sur nos données :

* Le modèle est un pipeline avec un CountVectorizer suivi d'un RandomForestClassifier.
* Le modèle est un pipeline avec un CountVectorizer suivi d'un choix de classificateurs le MultinomialNB.
* Le modèle est un pipeline avec un CountVectorizer suivi d'un choix de classificateurs le LogisticRegressionest .
* Le modèle est un pipeline avec un CountVectorizer suivi d'un choix de classificateurs le SVC.

Voici leur accuracy :

|  |  |  |  |
| --- | --- | --- | --- |
| RandomForest | Logistic Regression | Support Vector Machine (SVM) | Multinomial Naive Bayes |
| 89 ~ 91% | 91% | 92% | 89% |

Le plus grand accuracy obtenue est celle du modèle SVM

La classe DumpableModel fournit des méthodes pour sauvegarder et charger le modèle, en plus des méthodes standard d'entraînement et de prédiction.  
  
Pour finir nous avons adapté les commandes train et test(predict) pour qu’ils marchent. Pour cela on a split la base pour train et envoyé toute la base pour test(predict) cela indiquera 1 pour un film une chronique humoristique et 0 si ce n’est pas une chronique humoristique. La fonction predict créer un Csv avec les prédictions.

[**Partie 2 : Reconnaître**](https://github.com/afouchet/nlp_esgi/tree/main#partie-1-text-classification-prédire-si-la-vidéo-est-une-chronique-comique) **les noms de personnage dans le texte**

Nous avons dû faire face à une petite problématique car nous voulions pour voir appeler is\_name sans devoir supprimer ce qu’on a fait sur la première partis du TP pour cela nous avons modifier la pipelinne pour pouvoir appeler les deux situations avec les modèles approprié par exemple pour evaluation   
  
« python src/main.py predict --task=is\_comic\_video --input\_filename=data/raw/test.csv --model\_dump\_filename=models/model.json --output\_filename=data/processed/prediction.csv »  
  
si nous voulions avoir un fichier qui prédits si la vidéo est une chronique dans task nous mettions is\_comic\_video, si nous voulions avoir une liste de 0 et 1 qui mets 1 quand il s’agit d’un noms de personnage nous mettrons is\_name.

### Les modèles pour is name nous avons décidé de le construire comme ci-dessous

### **("dict\_vectorizer", DictVectorizer(sparse=True)) :** On convertit des dictionnaires de caractéristiques en vecteurs. Si les valeurs dans les dictionnaires sont des chaînes de caractères, elles seront converties en une matrice binaire de type "one-hot encoding". Si les valeurs sont des nombres, elles seront laissées telles quelles. Après l'aplatissement des données, il faut les convertir en un format qui peut être utilisé par un modèle de machine learning. DictVectorizer transforme les dictionnaires en vecteurs, ce qui est un format adapté pour la plupart des modèles de machine learning.

**"classifier" :** Cette étape du pipeline est le modèle de classification. Plusieurs classificateurs sont proposés :

* + **LogisticRegression :** Un modèle de régression logistique, qui est un classificateur linéaire. Il est efficace pour des espaces de caractéristiques de grande dimension et est utilisé ici avec un nombre maximal d'itérations de 1000.
  + **RandomForestClassifier :** Un classificateur basé sur un ensemble d'arbres de décision. Les forêts aléatoires sont puissantes et flexibles, mais peuvent être plus difficiles à interpréter et à optimiser.
  + **SVC :** Un classificateur à vecteurs de support, qui est un autre type de modèle linéaire. Il est souvent très performant, surtout avec des espaces de caractéristiques de grande dimension.
  + **("classifier", MultinomialNB())** : le Classificateur bayésien multinomial, on peut l’utilise car ce type de classificateur est bien adapté à notre situation car on a des compteurs ou des fréquences d'occurrence.

|  |  |  |  |
| --- | --- | --- | --- |
| **RandomForestClassifier** | Logistic Regression | **SVC** | Multinomial Naive Bayes |
| 96% | 96% | 96% | 95% |

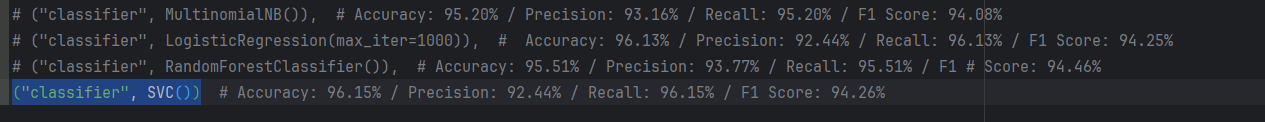
### Choix du Modèle

Si nous prenons le SVC nous avons remarqué qu’il ne prédit que la plupart du temps il y a que des 0 dans la liste, sur ce fait avoir une liste de 0 permet d’avoir une bonne accuracy la plupart du temps nous avons donc toruvais intéressent de faire une custome metric qui identifiera que les 1 et pas les 0 car cela va permettre de résoudre ce problème.

Nous avons donc cherché à trouvais la précision pour choisir notre modèle ci-dessous les résultats

|  |  |  |  |
| --- | --- | --- | --- |
| **RandomForestClassifier** | Logistic Regression | **SVC** | Multinomial Naive Bayes |
| 93.77% | 92.44% | 92.44% | 93.16% |

D’autres informations supplémentaires sur l’images de nos différents tests sur les modèles



Après les différents tests sur nos modèles nous avons choisi le RandomForestClassifier qui correspond mieux à ce que nous voulions qui a une plutôt bonne Accuracy et une bonne Précision.

1. Pour conclure

Cette structure permettra de les exécuter séquentiellement. Cela rend le processus de transformation des données et de classification plus propre et plus facile à gérer et à comprendre. Lorsque de nouvelles données sont introduites dans le pipeline, elles sont d'abord aplaties, puis converties en vecteurs, et enfin classifiées.

La partie du code crée un modèle de machine learning pour la classification de données structurées en listes de listes de dictionnaires, en utilisant un classificateur bayésien multinomial.

Pour les features nous avons fait, dans un premier temps la fonction

Cette fonction est utilisée pour diviser un texte en une liste de mots (tokens) en suivant des règles spécifiques. Elle gère différents cas pour s'assurer que la tokenisation est effectuée de manière à correspondre à la structure du texte.

Si le mot contient un tiret (mais ne commence ni ne se termine par un tiret), il est conservé tel quel.Si le mot se termine par deux points, il est également conservé tel quel.Si le mot contient une apostrophe, il est divisé en deux tokens à l'endroit de l'apostrophe.Dans tous les autres cas, le mot est conservé tel quel.

Une fonction preprocess\_text\_P2\_matching\_labels est conçue pour prétraiter du texte dans le contexte de la tâche "is\_name".

Cette fonction permet de convertir un texte brut en une liste de dictionnaires, où chaque dictionnaire contient des caractéristiques extraites d'un mot particulier dans le texte.

Le texte est divisé en mots (ou tokens) à l'aide de la fonction tokenize\_matching\_labels. Cette étape est importante car elle détermine comment le texte est segmenté et peut affecter la performance du modèle de machine learning.

Pour chaque mot dans le texte tokenisé, la fonction extrait des caractéristiques et les gardes dans un dictionnaire. Les caractéristiques extraites prennent un mot, sa position dans le texte, et ses mots voisins (précédant et suivant) pour extraire des caractéristiques pertinentes.

Tous les dictionnaires de caractéristiques sont rassemblés dans une liste, qui est renvoyée par la fonction.

Cela permet retour est une liste de dictionnaires, où chaque dictionnaire représente un mot du texte et contient des caractéristiques qui peuvent être utilisées pour la classification ou d'autres tâches de machine learning. Cette liste est passée notre modèle de machine learning, pour effectuer des prédictions ou des analyses.

Pour la main principale nous avons dû adapter notre code du a une erreur qu’on n’a pu apercevoir lors des prédictions cela prenait la première prédiction de chaque phrase ce qui créer des erreurs donc nous avons créé une fonction pour résoudre ce problème qui est « reshape\_predictions\_using\_position » cela permet de toujours d’avancer de 1 en ayant une position initiale. CELA permet d'aligner facilement les résultats du modèle avec les annotations originales, facilitant ainsi l'évaluation de la performance du modèle et l'identification des erreurs.

### -D’abord le prétraitement et tokenisation du texte

### Le système intègre une méthode de tokenisation qui prend en compte divers cas, tels que les tirets, les apostrophes et les deux points, afin de segmenter le texte en mots de manière précise. Cette étape cruciale garantit que la structure du texte est préservée, ce qui est essentiel pour l'alignement correct entre les mots et leurs étiquettes associées.

### -Puis l’extraction **de caractéristiques**

Au début cette partie se fait avec La fonction « extract\_word\_features\_P2 » mais nous l’avons directe intégrer dans « preprocess\_text\_P2\_matching\_labels » pour optimiser le code cela permet d’extraire des caractéristiques spécifiques de chaque mot, tels que sa longueur, sa position dans le texte, etc. Ces caractéristiques sont cruciales car elles peuvent aider notre modèle de machine learning à reconnaître des modèles spécifiques associés aux noms de personnages.

### -Ensuite le **modèle de machine learning**

Le pipeline de machine learning, défini dans le fichier main.py, est utilisé pour créer un modèle capable de classer les mots comme étant des noms de personnages ou non. Le pipeline inclut un transformateur pour aplatir les données, un DictVectorizer pour convertir les dictionnaires de caractéristiques en vecteurs, et un classificateur bayésien multinomial pour effectuer la classification.

### -Et pour finir l’entraînement **et la prédiction**

Le modèle est entraîné sur un ensemble de données où les noms de personnages sont marqués. Après l'entraînement, le modèle peut être utilisé pour prédire si les mots dans de nouveaux textes sont des noms de personnages ou non.

En résumé, l'ensemble de ces composants créer un système capable de reconnaître les noms de personnages dans le texte.

**Partie 3 : Assembler les modèles**

Pour cette dernière partie nous devions assembler les modèles, pour cela nous avons modifié les fichier main\_model, main et make\_features.

Pour commencer nous avons fait comme dans la partie deux nous avons séparer le pipeline. Cependant vu que nous avions déjà des modèles préparer, notre objectif était donc de pouvoir assembler les modèles pour cette partie.

Dans notre pipeline la fonction se concentre sur l'extraction de noms de comics à partir du DataFrame. Qui est affecté à la colonne "comic\_name" du DataFrame df. Comic\_name sera utilisée comme cible pour la formation d'un modèle de prédiction. Cependant nous avons eu un problème « no module name src » sur le make\_features. En effet on ne peut pas importer d’autre fonction, pour résoudre le problème nous avons mis les modèles en brute dans le main.py. Nous avons laissé en commentaire dans main\_model à la dernière ligne « COPY OF SMAE FUNCTION OF OTHER FILE DUE OF BUG ».

1. Construction pour l’assemblement des modèles

Nous avons mis en place un pipeline de prédiction en deux étape(P3). Pour cela on va d’abord faire **l’initialisation et chargement du modèle de prédiction "is\_comic\_video ». Nous avons utilisé un modèle random forest. Nous** commençons par initialiser une instance de DumpableModel, un modèle personnalisé conçu pour être sauvegardé et chargé le modèle spécifique pour la première tâche de classification, qui détermine si une vidéo est en lien avec un comique. Le model est chargé depuis le fichier models/model\_RDF.json.

**Puis nous avons extrait les caractéristiques et les prédictions de "is\_comic\_video"**. La fonction make\_features est appliquée au dataframe pour générer un ensemble de caractéristiques pertinentes à partir des données, adaptées à la tâche de classification "is\_comic\_video". Sur la base de ces caractéristiques, le modèle effectue des prédictions, qui sont ensuite ajoutées au dataframe original sous une nouvelle colonne result\_model\_is\_comic.

Seules les lignes du dataframe où la prédiction indique la présence d'une vidéo de comic (c'est-à-dire, result\_model\_is\_comic égal à 1) sont conservées pour les étapes suivantes.

Nous allons avoir le même procédé pour is\_comic\_video avec aussi le même modèle random forest.

**Chargement du modèle de prédiction "is\_name"** : Un nouveau modèle de DumpableModel est instancié et chargé avec un modèle différent qui est stocké dans models\_p2/model\_RDF.json. Ce modèle est conçu pour identifier les noms présents dans les données sélectionnées à l'étape précédente.

**Extraction de caractéristiques et prédiction "is\_name"** : La fonction make\_features est de nouveau utilisée pour transformer le dataframe filtré, cette fois pour répondre à la tâche "is\_name".

Les prédictions sont réorganisées pour correspondre à la structure des données d'origine grâce à la fonction reshape\_predictions\_using\_position. Ensuite, la fonction extract\_names est utilisée pour extraire les entités nommées à partir de la séquence de tokens initiale, en se basant sur les prédictions réorganisées.

La colonne comic\_name du dataframe est convertie en une liste Python, en évaluant chaque entrée comme une expression littérale. Le premier élément de chaque liste est extrait si la liste n'est pas vide, sinon « None » est inséré à la place.

En conclusion, la fonction P3 illustre une application de l'analyse prédictive en deux phases, où nos modèles de ML sont utilisés pour filtrer et extraire des informations spécifiques à partir d'un grand ensemble de données.

Nous obtenons une performance de 46% en accuracy après avoir tester les différentes compositions de modèle

1. Adaptation du test et de train

Pour adapter le test et train nous avons créé un autre fichier NER\_Model

Attention pour lancer le NER\_Model il faut mettre **if task == NER\_Model**

Voici des exemples de commande :

* python main.py test --task=NER\_MODEL -input\_filename=data/raw/train\_v2.csv --output\_filename=data/processed/prediction\_test.csv
* python main.py train --task=NER\_MODEL -input\_filename=data/raw/train\_v2.csv

Ce script contient une classe abstraite TextProcessor et une classe concrète NER\_Model qui hérite de TextProcessor. Cela permet de traiter le texte en utilisant le traitement du langage naturel pour nettoyer les données textuelles et extraire des entités nommées, spécifiquement des noms de personnes, à l'aide de la bibliothèque spaCy.

La classe TextProcessor permet de :

* **clean\_text(self, text: str) -> str:** méthode abstraite pour nettoyer le texte.
* **extract\_entities(self, text: str) -> List[str]:** méthode abstraite pour extraire des entités à partir du texte.
* **process\_dataframe(self, df: pd.DataFrame, text\_column: str) -> pd.DataFrame:** méthode abstraite pour traiter un DataFrame.

La classe NER\_Model implémente l'interface définie par TextProcessor et utilise le modèle de langue française de spaCy (fr\_core\_news\_sm) pour l'extraction d'entités nommées.

* **clean\_text(self, text: str) -> str:** Cette méthode nettoie le texte en supprimant la ponctuation et les espaces excédentaires.
* **extract\_entities(self, text: str) -> List[str]:** Cette méthode utilise spaCy pour identifier les entités nommées dans le texte et retourne une liste de noms de personnes (entités avec l'étiquette "PER").
* **process\_dataframe(self, df: pd.DataFrame, text\_column: str = 'video\_name') -> pd.DataFrame:** Cette méthode prend un DataFrame et applique les méthodes de nettoyage du texte et d'extraction d'entités à une colonne spécifiée, puis ajoute les résultats dans une nouvelle colonne NER\_result.

En plus des méthodes implémentées, NER\_Model contient une méthode supplémentaire find\_videos\_with\_name :

* **find\_videos\_with\_name(self, df: pd.DataFrame, name: str, text\_column: str = 'video\_name') -> pd.DataFrame:** Cette méthode filtre et retourne les lignes du DataFrame où un nom spécifique est trouvé dans la colonne des titres de vidéo. Elle utilise une recherche insensible à la casse pour voir si les titres de vidéo contiennent le nom donné.

Dans l'ensemble, ce script est structuré pour traiter et analyser du texte.

Cette structure permet d’affiche le résultat lors de l’utilisation de la commande train et quand on utilisera la commande test cela affichera le résultat et enregistrera dans un csv

-prediction\_p3.csv donne toutes les lignes avec les noms des comiques qui apparaisse

-prediction\_video\_by\_name\_p3.csv cela donnera toutes les lignes quand on donnera le nom de la personne exemple Laurence Bibot affichera les lignes qui contient Laurence Bibot

1. Conclusion

Nous avons construit une structure qui permet d’assemblée les 2 modèles pour pouvoir prédire si le nom d’un comique. Nous avons utilisé 2 modèles de random forest que nous avons loader. Puis nous enregistrons 2 fichiers csv un qui donnera tous les résultats de nos prédictions et une autre-ci on cible un comique pour voir dans quel filme il apparaît.