École Supérieure de Génie Informatique

TD 1

Auteur:
AL ASMI Alhusain.
CARMONE Alexandre.

Cours:
Traitement Automatique de langue Naturelle

Référent:
FOUCHET Arnaud

Contents

1 Introduction

Ce TD de Traitement Automatique du Langage Naturel (NLP), se penche sur un défi : la prédiction du caractère comique d'une vidéo en utilisant uniquement son titre.

Dans ce TD, nous explorons le potentiel des techniques NLP pour résoudre ce problème en utilisant Python et plusieurs bibliothèques spécialisées.

Le code que je présenterai dans ce rapport est un code modulaire et flexible permettra d'entraîner un modèle, de le sauvegarder, de réaliser des prédictions, et d'évaluer ses performances, le tout grâce à des commandes simples en ligne de commande.

2 Imports

Le code commence par plusieurs déclarations d'importation pour charger les bibliothèques et les fonctions nécessaires. Certaines de ces bibliothèques comprennent :

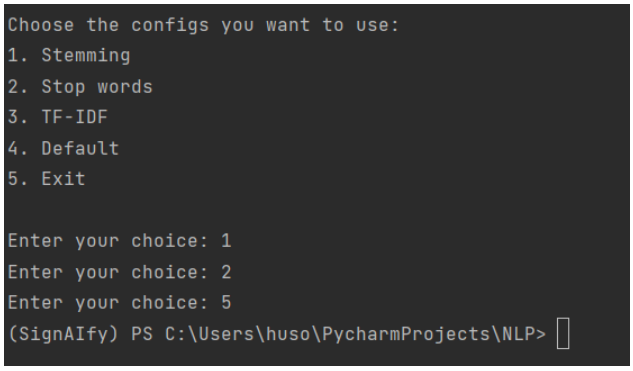
- **click :** Utilisé pour créer des interfaces en ligne de commande.
- **sklearn.model_selection :** Fournit des outils pour la sélection et l'évaluation de modèles, tels que la cross-validation.
- **pickle :** Utilisé pour sérialiser et désérialiser des objets Python, en particulier pour sauvegarder et charger des modèles entraînés
- **pandas :** Une bibliothèque populaire pour la manipulation des données et création des dataframes.
- Des modules personnalisés du package src, tels que make_dataset, make_features, make_model et dump.

3 Interface en Ligne de Commande (CLI)

La bibliothèque click est utilisée pour créer une interface en ligne de commande. Elle définit un groupe CLI nommé cli. Nous avons 3 commandes disponibles afin d'utiliser les 3 différentes fonctionnalités du code: **train, predict et evaluate**.

3.1 Train

Dans la fonction train, nous commençons par appeler la fonction make_model() pour créer un modèle et obtenir une configuration. Cette fonction lance un log dans la terminale permettant l'utilisateur de choisir les types de pré-traitement à appliquer sur les données.



Ces choix sont sauvegardés dans la liste de configurations et ensuite sauvegardée dans le même fichier Json que celui du modèle. Nous utilisons cette commande pour faire appel à la fonction train

```
python src/main.py train --task=is_comic_video --input_filename=src/data/raw/train.csv
--model_dump_filename=src/models/model.json
```

3.2 predict

La commande predict est définie avec des options pour la tâche, le nom du fichier d'entrée, le nom du fichier de sauvegarde du modèle et le nom du fichier de sortie. Cette commande est utilisée pour effectuer des prédictions avec le modèle entraîné.

Elle lit les données d'entrée à partir du fichier spécifié à l'aide de la fonction **make_dataset**. Ensuite, Elle charge un modèle pré-entraîné et une configuration à partir d'un fichier pickle sérialisé (model_dump_filename).

Similaire à la fonction train, elle applique les types de pré-traitement choisis et en suite les prédictions sont faites sur les titres. Une fois ceci est fait, les prédictions sont sauvegardées en tant que fichier CSV dans le répertoire "data/processed". Nous utilisons la commande suivante pour faire appel à la fonction predict.

```
python src/main.py predict --task=is_comic_video --input_filename=src/data/raw/test.csv
--model_dump_filename=src/models/model.json --output_filename=src/data/processed/prediction.cs
```

3.3 evaluate

La fonction evaluate réutilise les mêmes fonctions déjà mentionnées, la différence dans cette fonction c'est qu'elle effectue la validation croisée sur le modèle à l'aide de cross_val_score et affiche les scores et la moyanne des scores. La commande permettante d'appeler la fonction evaluate est la suivante:

```
python src/main.py evaluate --task=is_comic_video --input_filename=src/data/raw/train.csv
```

4 is_comic_video

4.1 Résultats

Après avoir testé 3 modèles en particuliers (Random Forest, Gradient Boosting Classifier, Ada Boost Classifier), et en utilisant différentes configurations de pré(traitement des données, chaque modèle a été soumis à une série de 8 tests distincts. Ces configurations comprenaient l'utilisation de count_vectorizer, stemming, stop words, et TF-IDF, ainsi que diverses combinaisons de ces techniques

Pour déterminer quels modèles sont meilleurs, nous pouvons examiner les moyennes des scores (Moyenne des scores) pour chaque type de modèle et configuration de pré-traitement des données. Plus la moyenne est élevée, meilleure est la performance du modèle. Voici une évaluation des meilleurs modèles en fonction des moyennes des scores :

4.1.1 Random Forest

- Configuration : TF-IDF
- Moyenne des scores : 0.9124

4.1.2 Gradient Boosting Classifier

- Configuration : count_vectorizer, stemming
- Moyenne des scores : 0.9249

4.1.3 Ada Boost Classifier

- Configuration : count_vectorizer, stemming
- Moyenne des scores : 0.9174

En fonction des moyennes des scores, le modèle Gradient Boosting Classifier avec la configuration count_vectorizer et stemming (Modèle 2) semble être le meilleur parmi les trois types de modèles testés. Il est suivi de près par le modèle Ada Boost Classifier avec la même configuration (Modèle 2). Le modèle Random Forest avec la configuration TF-IDF (Modèle 4) obtient également de bonnes performances.

Pour des informations plus détaillées sur les performances de chaque modèle, y compris les résultats de chacun des 8 tests, il suffit de consulter le répertoire "Resultats". Vous pouvez retrouver un fichier dédié à chaque modèle.

5 is_name

5.1 Make_features

la section « is_name » est une partie centrale de la procédure de traitement des données pour la tâche spécifique de classification des noms, c'est la NER(name entity recognition). Cette section est conçue pour préparer les caractéristiques et les étiquettes nécessaires à l'entraînement d'un modèle d'apprentissage automatique. Elle implique une série d'étapes visant à transformer les données brutes en un format approprié pour l'analyse et la prédiction ou l'usage de modèle plus complexe comme BERT ou GPT.

Pour la tâche « is_name », la fonction se concentre sur la transformation des données textuelles tokenisées. Elle associe chaque mot à son étiquette respective et extrait des caractéristiques pertinentes de chaque mot. Par exemple, elle évalue si un mot est le dernier ou le premier d'une phrase, ainsi que s'il est écrit avec une majuscule. Ces caractéristiques sont organisées en une structure de données adaptée à l'entraînement d'un modèle de classification des noms. La section "is_name" du script adopte une approche particulière en associant chaque mot individuel à son étiquette correspondante, plutôt que d'associer des phrases entières avec des vecteurs de labels. Cette approche se concentre sur une granularité fine, traitant chaque mot dans le contexte de la tâche de classification des noms.

5.2 Résultats

- Model 1
  - Model: Random Forest
  - Model used: Random Forest
  - Scores: 0.9563829, 0.95, 0.950505588, 0.94944119, 0.948908994
  - Average scores: 0.9510477506142923
- Model 2
  - Model: Ada Boost Classifier
  - Model used: Ada Boost Classifier
  - Scores: 0.9526595, 0.95372340, 0.9526343799893561, 0.9505055, 0.95263437
  - Average scores: 0.9524314653561763
- Model 3
  - Model: Gradient Boosting Classifier
  - Model used: Gradient Boosting Classifier
  - Scores: 0.9526595, 0.9537234, 0.9526343, 0.9521021, 0.95263437
  - Average scores: 0.9527507841427649
- Model 4
  - Model: MLP Classifier

  - Model used: MLP Classifier
  - Scores: 0.955851, 0.95, 0.9494411, 0.94678020, 0.9489089
  - Average scores: 0.9501962904668622

- Model 5
  - Model: Logistic Regression
  - Model used: Logistic Regression
  - Scores: 0.952659, 0.9537234, 0.95263437, 0.95263437, 0.95263437
  - Average scores: 0.9528572237382946

- Model 6
  - Model: Support Vector Classifier
  - Model used: Support Vector Classifier
  - Scores: 0.9547872, 0.9537234, 0.9531665, 0.951569984, 0.9531665
  - Average scores: 0.9532827556531881

- Model 7
  - Model: Stochastic Gradient Descent Classifier
  - Model used: Stochastic Gradient Descent Classifier
  - Scores: 0.955851063, 0.9531914893, 0.949973390, 0.94784459, 0.95050558
  - Average scores: 0.9514732259123798

- Model 8
  - Model: Voting Classifier (Random Forest, Logistic Regression, Gradient Boosting Classifier)
  - Model used: Voting Classifier
  - Scores: 0.952659574, 0.95372340, 0.95263437, 0.952102182, 0.952634379
  - Average scores: 0.9527507841427649

- Model 9
  - Model: Bagging Classifier
  - Model used: Bagging Classifier
  - Scores: 0.954255319, 0.94787234, 0.94944119, 0.94944119, 0.951037
  - Average scores: 0.9504095659755641

- Model 10
  - Model: Stacking Classifier
  - Model used: Stacking Classifier (Random Forest + MLP Classifier + Logistic Regression)
  - Scores: 0.95638297, 0.9521276, 0.949973390, 0.94837679, 0.951569984

- Average scores: 0.9516861617202451

Les différents modèles ont été évalués selon leurs performances à l'aide des scores obtenus. Après l'analyse, il est observé que le modèle "Random Forest" a démontré la meilleure performance avec un score moyen de 0.9510477506142923. Ce modèle a été suivi de près par d'autres modèles tels que "Gradient Boosting Classifier" avec un score moyen de 0.9527507841427649 et "Logistic Regression" avec un score moyen de 0.9528572237382946. Cependant, après comparaison, le modèle ayant obtenu le meilleur score moyen était le "Support Vector Classifier" avec un score moyen de 0.9532827556531881.

On aurait pu utiliser le POS-Tagger de nltk pour récupéré tous les noms propre et filtrer ainsi la phrase. Cela nous aurait permis d'avoir un modèle qui détermine uniquement si le nom propre est associé à un nom de comics.

Le modèle CamemBERT nous permet de faire tout type de classification sur du texte comme la name entity recognition il existe pour cela le modèle camembert-ner (https://huggingface.co/Jean-Baptiste/camembert-ner) on peut directement utiliser le modèle comme POS-tagger de nltk et ensuite prédire si l'entité est associé à un comics. Ou on peut fine tuner le modèle pour qu'il prédise uniquement si le mot est un nom de comics.