Traitement Automatique de Langage

Rapport de Projet

Equipe:
ALLEMAND Fabien
LEBOT Samuel

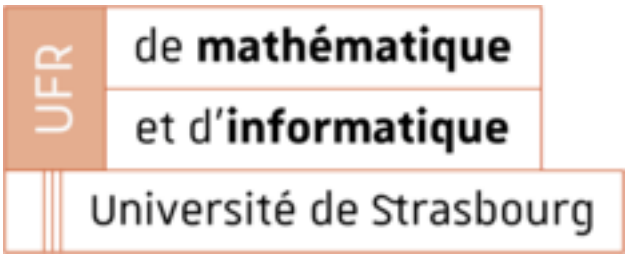


Table des matières

Liste des figures

```
train_data["genre"].value_counts()
```
Python

```
drame              501
comédie            483
romance            443
policier           331
horreur            299
science fiction    298
biopic             191
documentaire       167
historique         162
Name: genre, dtype: int64
```

Figure 1: Effectifs des classes dans les données d'entraînement

Introduction

L'objectif du projet est de réaliser un système de recherche d'information dans une collection de descriptions de films publiées sur Allociné.
Le projet se décompose en deux parties:

- Prédiction du genre des films par TAL
- Visualisation des résultats

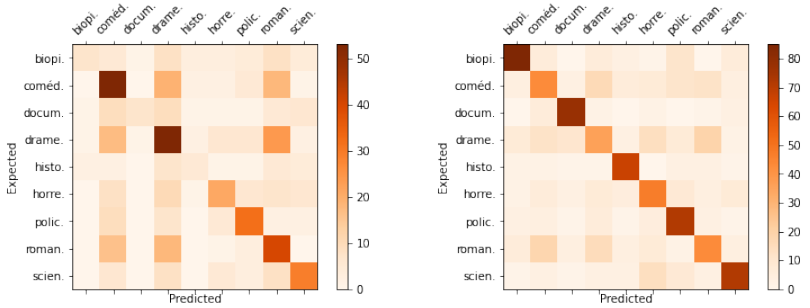
1 Analyse et Pré-traitement des Données

Dans un premier temps, les données d'entraînement (*allocine_genres_train.csv*) peuvent être chargées grâce à la fonction read_csv de la bibliotèque Pandas [5] en précisant le séparateur (sep=","). L'utilisation des méthodes head, tail, describe, info et hist permettent de visualiser et comprendre les données contenues dans le jeu de données complet.

Dans le cadre de ce projet seules les données contenues dans les colonnes *titre* et *synopsis* seront utilisées pour déduire la valeur contenue dans *genre*. Le jeu de données peut être réduit à ces trois features.
Etant donné que les données vont être utilisées pour de l'apprentissage automatique, il faut vérifier s'il y a des valeurs manquantes. Les méthodes isna et sum ne révèlent aucune valeur manquante dans les données d'entraînement.
La proportion des classes dans le jeu de données peut avoir un impact sur l'apprentissage. Les classes ayant un effectif plus faible seront généralement moins bien "apprises". La figure 1 montre que les classes n'ont pas toutes la même proportion dans le jeu de données: il y a beaucoup d'individus de la catégorie *drame* alors que les classes *biopic*, *documentaire* et *historique* sont très peu représentées.

Après de nombreuses expériences, il s'avère que rééquilibrer les classes par *oversampling*, c'est à dire: dupliquer des individus des classes les moins représentées, donne de meilleurs résultats quelque soit la méthode utilisée. (Figure 2)
Dans tout la suite, les résultats présentés correpondront aux résultats obtenus avec le jeu de données d'entraînement rééquilibré par *oversampling* à l'aide de l'objet RandomOverSampler de la bibliotèque Imbalanced-learn [3].

Remarque | Le jeu de données d'entraînement contient trop peu de données pour effectuer un équilibrage des classes par *undersampling*, c'est à dire: supprimer des individus des classes les plus représentées. Les résultats obtenus avec cette méthode étaient généralement moins bons que sans équilibrage.



(a) Matrice de confusion du modèle de régression logistique avec des classes déséquilibrées

(b) Matrice de confusion du modèle de régression logistique avec des classes équilibrées par oversampling

Figure 2: Comparaison des matrices de confusion pour des données déséquilibrées et équilibrées: les classes les moins représentées dans le jeu de données d'entraînement sont moins bien apprises lorsque les classes sont déséquilibrées.

```python
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

column_trans = ColumnTransformer(
    [
        # Titre: tf-idf
        ("titre_tfidf", titre_vectorizer, "titre"),
        # Synopsis: tf-idf
        ("synopsis_tfidf", synopsis_vectorizer, "synopsis"),
        # Synopsis: statistiques
        (
            "synopsis_stats",
            Pipeline(
                [
                    ("text_stats", text_stats_transformer),
                    ("vect", text_stats_vectorizer),
                    ("scaling", min_max_scaler)
                ]
            ),
            "synopsis"
        )
    ],
    # Others
    remainder="passthrough"
)
```

✓ 0.1s Python

Figure 3: Pipeline de pré-traitement des données

2 Méthodes Basiques

2.1 Pré-traitement des Données

Il est possible de mettre en place une pipeline (Figure 3) afin de vectoriser les données d'apprentissage et d'en extraire des informations statistiques.
Dans cette pipeline, les données déjà tokénisées vont être vectorisées selon la méthode TF-IDF en supprimant les stop-words. Puis des objets FUnctiontransformer et DictVectorizer de la bibliotèque Scikit-learn [6] vont être utilisés pour obtenir les valeurs statistiques telles que la longeur du synopsis en nombre de mot et en nombre de phrases. Un MinMaxScaler est utilisé pour normaliser ces données afin qu'elles aient le même poids lors de l'apprentissage.

2.2 Apprentissage Simple

Avant tout apprentissage supervisé, il faut définir un jeu d'apprentissage et un jeu de test avec par exemple la méthode train_test_split de Scikit-Learn en spécifiant la proportion du jeu de données sélectionnée pour les données de test (typiquement: test_size=0.2) et précisant shuffle=True pour que les données ne soient pas

```python
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LogisticRegression

# Pipeline pré-traitement et apprentissage
classifier_pipeline = make_pipeline(
    # Préparation des données pour l'apprentissage
    column_trans,
    # Algorithme d'apprentissage
    LogisticRegression()
)
```

✓ 0.0s Python

```python
# Apprentissage avec les données d'entraînement
classifier_pipeline.fit(X_train, y_train.to_numpy())
```

✓ 1.5s Python

Figure 4: Pipeline d'entraîenement d'un modèle de régression logistique

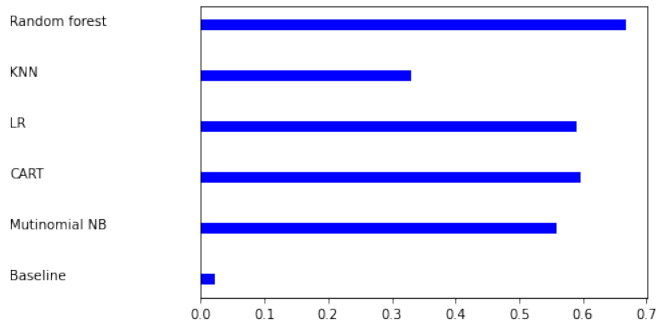


Figure 5: Précision de différents algorithmes lors d'un apprentissage par validation croisée avec 5 plis

sélectionnées séquentiellement (ce qui pourrait impacter le résultat si le jeu de données est trié par classes).

Remarque | | Le jeu de données *allocine_genres_test.csv* correspond au jeu de validation qui ne sera utilisé qu'après avoir définitivement choisi la méthode de prédiction.

Il est alors possible de créer une nouvelle pipeline pour automatiser le pré-traitement et l'apprentissage (Figure 4).

On peut ensuite faire des prédictions sur le jeu de test avec la méthode predict pour évaluer le modèle. Ce modèle de régression logistique donne une précision de 0.6 et un rappel de 0.61 (donc un score f1 de 0.6).

2.3 Apprentissage par Plis et Comparaison des Algorithmes

Comme vu précédemment, la taille et les individus sélectionnés pour entrainer le modèle peuevent avoir une influence sur la qualité de l'apprentissage. Pour contrer ce biais on peut utiliser des plis pour l'apprentissage: au lieu de diviser les données d'entraînement en un jeu d'entrainement et un jeu de test, on peut diviser les données d'entraînements en plusieurs plis qui seront tour à tour des données d'entraînement ou de test. Celà permet d'obtenir une idée de la performance moyenne du classifieur. On utilise la méthode StratifiedKFold de Scikit-Learn qui a la particularité de conserver la proportion d'individus de chaque classe dans les plis.

En utilisant cette méthode, on peut comparer le modèle de régression logistique avec d'autres algorithmes. Les meilleurs résultats proviennent d'un modèle de forêt aléatoire (précision proche de 0.7). (Figure 5)

3 Réseau Neuronal Convolutif

Les réseaux neronaux convolutifs (ou Convolutional Neural Network)[2] sont des réseaux de neurones qui utilisent des opérations de réduction de dimension et de convolution afin d'extraire des caractéristiques sur les données.

```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, None)]            0

embedding (Embedding)        (None, None, 700)         5600000

conv1d (Conv1D)              (None, None, 64)          224064

max_pooling1d (MaxPooling1D  (None, None, 64)          0
)

conv1d_1 (Conv1D)            (None, None, 64)          20544

global_max_pooling1d (Globa  (None, 64)                0
lMaxPooling1D)

dense (Dense)                (None, 64)                4160

dropout (Dropout)            (None, 64)                0

dense_1 (Dense)              (None, 9)                 585

=================================================================
Total params: 5,849,353
Trainable params: 5,849,353
Non-trainable params: 0
```

Figure 6: Architecture du réseau neuronal convolutif

3.1 Pré-traitement des Données

Ces réseaux de neurones sont généralement associés à un réseau dense pour efféctuer la prédiction après avoir extrait les informations des données. Dans le cadre du projet, la prédiction correspond à une classe (genre du film) parmi neuf à l'aide du titre et du synopsis du film. La dernière couche du réseau comport donc neuf cellules qui donneront chacune la probabilité que le film appartienne à chaque classe. Pour l'apprentissage et la prédiction, il faut lister tous les genres, leur associer un indice et remplacer les valeurs dans les données (voir dictionnaires genre2id et id2genre).
Dans le dataframe, les colonnes *titre* et *synopsis* doivent être fusionnées afin de pouvoir les vectoriser dans la suite.

3.2 Indexation du Vocabulaire

Les réseaux neronaux convolutifs effectuent des opérations sur des données numériques. Pour une application en TAL, il faut transformer le contenu sous forme de texte en données numériques, par exemple représenter chaque mot comme un vecteur de nombres réels dans un espace de grande dimension.
Dans le programme, l'objet de type TextVectorization de la bibliothèque Tensorflow [7] permet d'effectuer la vectorisation, chaque mot correpond alors à son indice dans le vecteur. Par exemple: si TextVectorization place le mot *bonjour* à la troisième position du vecteur alors le mot *bonjour* sera traité par le CNN comme la donnée 2.

3.3 Chargement de Plongements de Mots Pré-entraînés

La vectorisation des mots du vocabulaire est une opération complexe mais il est possible d'utiliser des plongements pré-entraînés.
Dans ce projet, le plongement *frWiki_no_phrase_no_postag_700_cbow_cut100.bin* [4] de dimension 700 a été utilisé pour vectoriser les données.

3.4 Construction et Entraînement du Réseau Neuronal Convolutif

Un CNN est composé de plusieurs couches effectuant différentes opérations (réduction de dimension, convolution, couches denses). Le réseau utilisé dans ce projet contient seulement deux couches de réduction de dimension et deux couches de convolution (Figure 6).

Remarque | Après plusieurs expériences avec différentes architectures, il semblerait qu'ajouter des couches de réduction de dimension et de convolution n'améliorent pas la qualité des résultats dans le cadre de cette étude.

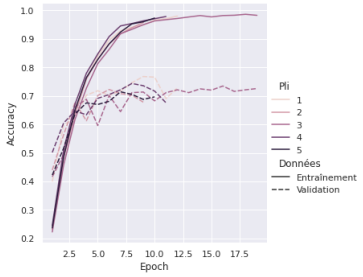


Figure 7: Précision du réseau neuronal convolutif au cours de l'entraînement

Les opérations de convolution du CNN sont des opérations qui doivent être apprises au cours de l'entraînement. [1] La taille des batchs est de 64 et le nombre d'epochs choisi est 25, permettant un apprentissage plus précis mais aussi plus long. Pour contrer cela, un callback de type EarlyStopping a été ajouté (arrêt de l'apprentissage lorsque la précision du modèle diminue).

Conclusion

Bibliographie

[1] Cnn et couche de convolution. https://inside-machinelearning.com/cnn-couche-de-convolution/. Accessed: 2023-04-30.

[2] Convolutional neural networks. https://www.ibm.com/topics/convolutional-neural-networks. Accessed: 2023-04-30.

[3] Imbalanced-learn. https://imbalanced-learn.org/stable/. Accessed: 2023-04-30.

[4] jean-philippe fauconnier. http://fauconnier.github.io/. Accessed: 2023-04-30.

[5] Pandas. https://pandas.pydata.org/. Accessed: 2023-04-30.

[6] Scikit-learn. https://scikit-learn.org/stable/. Accessed: 2023-04-30.

[7] Tensorflow. https://www.tensorflow.org/?hl=fr. Accessed: 2023-04-30.