NLP (Natural Language Processing)

Kevin et Mickaël

Contexte du projet

En règle générale, le nombre d'avis sur un film peut être important et par conséquent le temps de lecture de chaque commentaire peut être une tâche lourde. Alors comment **déterminer de manière rapide si un film a eu du succès auprès des spectateurs (ou pas)** ? Dans ce contexte, l'idée du projet est d'utiliser des algorithmes d'apprentissage automatique pour la tâche d'analyse de sentiment des spectateurs via leur critique.

Tout d'abord, il sera question de **récupérer les données directement du site d'Allociné**. En d'autres termes, nous allons **scraper** les pages qui nous intéressent sur ce site à savoir les critiques des personnes pour le film **Inception** et **Sonic 2**.

En naviguant sur la page des critiques, vous vous apercevrez que seules deux types d'information ici nous intéressent : **la note du spectateur ainsi que son avis**. Pourquoi la note ? Parce que nous allons **entraîner un modèle de type supervisé et plus précisément un classifieur** et donc la note va nous aider à récupérer la classe pour étiqueter le commentaire. Pour cela, nous considérons qu'une note au-dessus de 3 est considérée comme satisfaisante. Sinon, l'avis est négatif. Ici, nous avons donc réduit le problème à une classification binaire.

Etape 1 : Web Scraping des données d'avis de spectateurs

```
urls_pages=[["https://www.allocine.fr/film/fichefilm-281203/critiques/spectateurs/?page=",14],  # SONIC
            ["https://www.allocine.fr/film/fichefilm-143692/critiques/spectateurs/?page=",480]] # Inception

for film in urls_pages:
    for page in range(1,film[1]):
        url = f"{film[0]}{page}"
        reponse=requests.get(url)
        soup = BeautifulSoup(reponse.text,"html.parser")
        donnees=soup.find_all("div", {"class":"hred review-card cf"})

        for comment in donnees:
            id_com = comment.get("id").split("_")[1]
            note= comment.find("span",{"class":"stareval-note"}).text.replace(",",".")
            commentaire = comment.find("div",{"class":"content-txt review-card-content"}).text.replace("'"," ").replace("\n","")

            with open('data2.csv','a',newline='') as fichiercsv:
                writer=csv.writer(fichiercsv)
                writer.writerow([id_com,commentaire,note])

            query =f"""INSERT INTO allo_cine(id_comment, commentaire, note) """
            query += f"""VALUES ('{id_com}','{commentaire}',{note})"""
            query += f"""ON DUPLICATE KEY UPDATE id_comment='{id_com}' """
            cursor.execute(query)
        bdd.commit()
```

Etape 2 : Préparation des données

```
df = pd.read_csv("data2.csv", names=["id","commentaire","note"])
```

Transformation du csv en dataframe

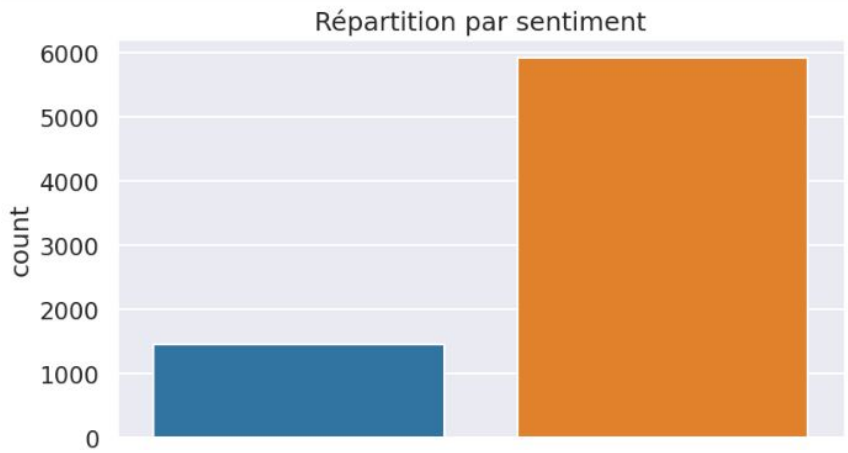
```
df.isna().sum()
```

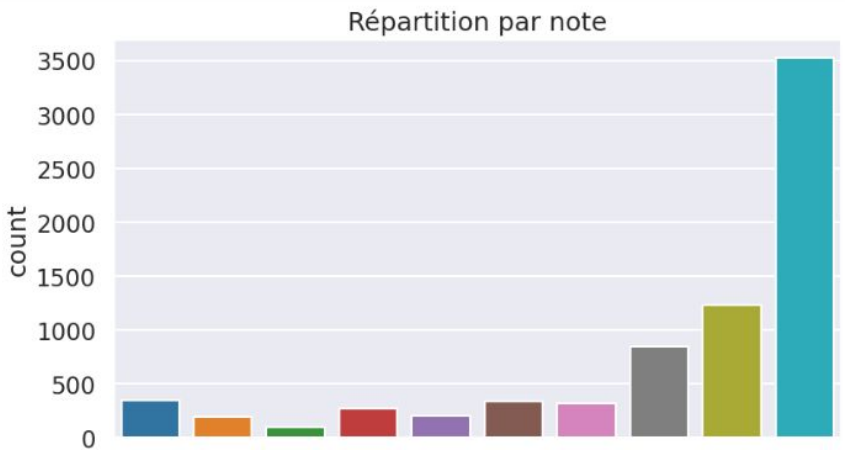
```
df.dropna(inplace=True)
```

Suppression des commentaires vides

```
df.loc[(df.note>3),'sentiment']=1
df.loc[(df.note<=3),'sentiment']=0
```

Création d'une colonne sentiment pour définir un commentaire positif ou non.





Etape 3: Split du dataframe et équilibrage des données

```
X_train, X_test, y_train, y_test = train_test_split(df.commentaire, df.sentiment, test_size=0.2, random_state=42)
print(y_train.value_counts())
```

```
1.0    4732
0.0    1159
```

Il y a 3 fois plus de commentaires négatifs dans notre jeu d'entraînement. Il va donc falloir utiliser la data augmentation pour l'équilibrer.

La méthode de data augmentation choisie est la back translation. Nous allons dans un premier temps traduire tous les commentaires négatifs en anglais puis revenir en français ce qui permet de créer des synonymes. Puis on va refaire la même étape avec une traduction en allemand.

```
def back_translate(sequence, langue):
    translator = Translator()
    #translate to new language and back to original
    translated = translator.translate(sequence, dest = langue).text
    #translate back to original language
    translated_back = translator.translate(translated, dest = 'fr').text
    return translated_back

count= X_train[y_train==0].shape[0]*3
langue=['en','nl']
for i in langue:
    for commentaire in X_train[y_train == 0]:
        output_translate = back_translate(commentaire, i)
        X_train = pd.concat([X_train,pd.Series(output_translate)])
        y_train = pd.concat([y_train,pd.Series(0)])
        print(count)
        count-=1
```

```
1.0    4732
0.0    4636
```

Le jeu d'entraînement étant bien équilibré nous pouvons passer au traitement des données...

Etape 4: Standardisation et Lemmatisation

La standardisation consiste a remplacer tous les caractères qui ne sont pas utiles voir néfastes pour la suite de la création de notre modèle de prédiction.

La lemmatisation d'un verbe ramène ce verbe à l'infinitif et pour les autres mots, la lemmatisation ramène ce mot au masculin singulier. Cette fonction sera appelé lors de la vectorisation

```python
def standardize_text(donnees):
    donnees = donnees.str.replace(r"http\S+", "",regex=True)
    donnees = donnees.str.replace(r"http", "",regex=True)
    donnees = donnees.str.replace(r"@\S+", "",regex=True)
    donnees = donnees.str.replace(r"[0-9(),;!:?@<>.=\'\`\"\-\_\n]
    donnees = donnees.str.replace(r"@", "at",regex=True)
    donnees = donnees.str.replace("é", "e")
    donnees = donnees.str.replace("è", "e")
    donnees = donnees.str.lower()
    return donnees

X_train = standardize_text(X_train)
```

```python
liste_stopwords=stopwords.words('french')
def tokenisation(donnees,liste_stopwords):
    lemmatizer = FrenchLefffLemmatizer()
    corpus = []
    exclusion = stopwords.words('french')+liste_stopwords

    for i in range(0, len(donnees)):
        message = donnees.iloc[i]
        message = message.split()
        message =[word for word in message if word not in exclusion]
        message = [lemmatizer.lemmatize(word) for word in message]
        message = ' '.join(message)
        corpus.append(message)

    return corpus
```

Etape 5: Vectorisation

L'apprentissage *machine* exploitant des variables numériques, il est nécessaire de convertir un *texte* à analyser en une représentation vectorielle.

Il existe plusieurs modèles de vectorisation : TfidVectorizer, CountVercorizer, HashingVectorizer, Word2vec

Etape 6: Entraînement d'un modèle

Pour prédire un commentaire nous allons entraîner un modèle, là encore nous disposons de plusieurs modèles : LogisiticRegression, MultinomialNB...

Conclusion étape 5 et 6

Le meilleur score a été obtenu avec le vecteur TfidVectorizer et le modèle MultinomialNB pour un résultat de 88,25% de bonnes prédictions sur le jeu de test et sans recherche d'hyperparamètres.

Etape 7 : Amélioration du modèle

```
param_grid = {'alpha': np.arange(0.001, 1, 0.05)}

model_selection ={"StatifiedKFold": StratifiedKFold(4),
                  "Kfold": KFold(5, random_state=42, shuffle=True),
                  "ShuffleSplit": ShuffleSplit(4, test_size=0.2)}

for v in model_selection.values():
    grid = GridSearchCV(MultinomialNB(), param_grid, cv=v)
    grid.fit(X_train_vec, y_train)
    print("Le meilleur score avec ",str(v).split("(",1)[0],"est",grid.best_score_,"avec les parametres",
    grid.best_params_)
```

```
meilleur score avec  StratifiedKFold est 0.9155636208368916 avec les parametres {'alpha': 0.3010000000000
meilleur score avec  KFold est 0.9181242061970334 avec les parametres {'alpha': 0.101}
meilleur score avec  ShuffleSplit est 0.9212913553895412 avec les parametres {'alpha': 0.1510000000000000
```

```
model = grid.best_estimator_
model.fit(X_train_vec, y_train)
y_pred = model.predict(X_test_vec)

print("L'accuaracy score est de :", accurac
cm = confusion_matrix(y_pred, y_test)
sns.heatmap(cm, annot=True, fmt='d')
plt.show()
```

```
ccuaracy score est de : 0.8710115410726409
```

Etape 8 : Exportation du modèle avec Jolib

```
model_complet={ "Xtrain_translate":X_train,
                "ytrain_translate":y_train,
                "vector": vectorizer,
                "model":MultNB,
                }

joblib.dump(model_complet, 'brief_nlp_joblib.joblib')
```

Ici on exporte aussi le X_train, y_train pour ne pas refaire la back translation si on revient plus tard sur le projet.

Etape 9 : Wordcloud

Pour réaliser le wordcloud nous avons augmenté les stopwords en y ajoutant tous les mots qui étaient communs au commentaire négatif et positif. Nous nous sommes servi d'une image en png en mask pour que notre image de wordcloud ressemble à un pouce.

```
mask = np.array(Image.open("app/static/images/p_vert.png"))
mask[mask == 1] = 255
print("Wordcloud sur les commentaires positif")
wordcloud_pos = WordCloud(background_color = 'white', mask=mask,
 max_words = 100,min_word_length=3).generate(" ".join(tokenisation(X_train[y_train == 1], liste_stopwords))+" ")
plt.imshow(wordcloud_pos)
plt.axis("off")
plt.savefig("app/static/images/p__words_vert.png")
plt.show()
```

Etape 10: Réalisation d'une application Flask

Dans le fichier util.py:

On importe notre modèle de machine learning, le lemmatiseur et les stopwords.

```
import joblib
from nltk.corpus import stopwords
from french_lefff_lemmatizer.french_lefff_lemmatizer import FrenchLefffLemmatizer

vector = joblib.load('brief_nlp_joblib.joblib')['vector']
LReg = joblib.load('brief_nlp_joblib.joblib')['model']
```

Etape 10: Réalisation d'une application Flask

```
def standardize_phrase(donnees):
        donnees = donnees.replace(r"http\S+", "")
        donnees = donnees.replace(r"http", "")
        donnees = donnees.replace(r"@\S+", "")
        donnees = donnees.replace(r"[0-9(),;!:?@<>.=\'\`\"\-\_\n]", " ")
        donnees = donnees.replace(r"@", "at")
        donnees = donnees.replace("é", "e")
        donnees = donnees.replace("è", "e")
        donnees = donnees.lower()

        lemmatizer = FrenchLefffLemmatizer()
        corpus = []
        message = donnees.split()
        message =[word for word in message if word not in stopwords.words('french')]
        message = [lemmatizer.lemmatize(word) for word in message]
        message = ' '.join(message)
        corpus.append(message)

        return corpus

def predict_com(corpus_phrase):
    vectorisation=vector.transform(corpus_phrase)
    prediction = LReg.predict(vectorisation)
    return prediction
```

Dans le fichier util.py:

Les fonctions qui permettent de standardiser le commentaire envoyé, et de faire la prédiction du commentaire standardisé.

Etape 10: Réalisation d'une application Flask

Dans le fichier routes.py:

La méthode GET qui envoie notre page.

La méthode POST qui réceptionne le commentaire. Le commentaire est renvoyé, ainsi que la réponse du modèle sous forme de booléen.

```python
from flask import render_template, redirect, request, flash
from app import app

app.secret_key = b'_5#y2L"F4Q8z\n\xec]/'

from util import standardize_phrase, predict_com

@app.route('/', methods=['GET', 'POST'])
def textarea():
    if request.method == 'GET':
        return render_template('base.html')

    elif request.method == 'POST':
        commentaire = request.form['commentaire']
        bool = ''

        if len(commentaire) == 0:
            flash('Votre commentaire est trop court !!')
        else:
            standard = standardize_phrase(commentaire)
            predict = predict_com(standard)

            if predict == 0:
                bool =  False
            elif predict == 1:
                bool =  True

        return render_template('base.html', commentaire = commentaire, like = bool)
```

Etape 10: Réalisation d'une application Flask

```
<main>
  {% if commentaire %}
  <div class="response-IA">
    <div class="container-com">
      <div class="guillemet">&#10077</div>
      <p class="commentaire">{{commentaire}}</p>
      <div class="guillemet">&#10078</div>
    </div>
    {% if like %}
    <div class="response like"></div>
    {% elif like==False %}
    <div class="response dislike"></div>
    {% endif %}
    <a href="/"><button>Tester un autre commentaire</button></a>
  </div>
  {% else %}
  <div class="textarea">
    <div class="container-form">
      <form action="" method="POST">
        <label for="commentaire">
          <textarea name="commentaire" id="commentaire"></textarea>
        </label>
        <input type="submit" value="tester mon commentaire" />
      </form>
      <div class="flash-message">
        {% with messages = get_flashed_messages() %}
        {% if messages %}
        {% for message in messages %}
        <p>{{ message }}</p>
        {% endfor %}
        {% endif %}
        {% endwith %}
      </div>
      <div class="container-avis">
      <div class="avis like"></div>
      <div class="avis dislike"></div>
    </div>
  </div>
  </div>
  {% endif %}
</main>
```

Plusieurs conditions gèrent ce qui apparaît dans l'application. Ainsi, la première condition gère l'apparition du textarea ou de la réponse, suivant l'absence ou la présence d'un commentaire. La condition présente dans la réponse permet de gérer l'image de pouce.

```
<body class="{{like}}">
```

La réponse du modèle est transformée sous forme de class qui fait apparaître l'animation correspondante à une réponse positive ou négative.

Démonstration .....