François Labastie

Cas Pratique E2

Améliorer une application de modération de texte grâce à l'intelligence artificielle

Décembre 2020

Titre professionnel "Développeur en intelligence artificielle"

de niveau 6 enregistré au RNCP sous le n°34757

Passage par la voie de la formation -

parcours de 19 mois achevé le 20 octobre 2020

Introduction · Problématique business & définition du besoin client

La cinémathèque française propose divers blogs : http://blog.cinematheque.fr/ ainsi qu'un espace de commentaires. Afin de prévenir une éventuelle vague de commentaires haineux, et pour se soumettre à la législation qui oblige une modération sur les blogs, cet organisme souhaite mettre en place un système de contrôle automatisé. En effet, les éditeurs de services qui ne prennent pas de mesures préventives sérieuses s'exposent à des risques d'amende[1], voire de fermeture du site.

En tant que développeur en intelligence artificielle, et pour répondre à la demande de ce client, nous sommes chargés de créer un système de modération à mettre en place sur ces blogs. La nouvelle solution proposée fera appel à une API de modération avec intelligence artificielle.

Afin de présenter et analyser notre nouvelle solution nous créons - à l'aide du framework Python Flask[2] - un site prototype[3] avec formulaire de saisie de commentaires. Ce prototype nous permet de tester à la fois la solution actuelle du client puis la comparer avec notre nouvelle proposition d'IA. Lié à une base de données SQLite, ce prototype présente un formulaire de saisie de texte et affiche le résultat de la modération automatique. Les commentaires valides sont enregistrés en base de données, et les cinq derniers sont affichés sur la page.

Les analyses présentées à notre client sont basées sur l'architecture et le code de ce site prototype qui est déployé localement ainsi que sous forme de conteneur Docker sur la plateforme Azure, le temps des évaluations et tests.

[1] https://www.cnil.fr/fr/definition/sanction
[2] https://flask.palletsprojects.com/en/1.1.x/
[3] https://github.com/flabastie/text_moderation

1. Application existante

1.1 Détection par comparaison de listes

Sur les blogs, le système de modération actuel repose sur une liste de mots prohibés, stockée dans un fichier texte, et mise à jour manuellement. Lorsqu'un internaute propose un commentaire, celui-ci est comparé avec la liste des mots à bannir. Le commentaire - chaîne de caractère - est *tokenisé*[4] et transformé en liste. La présence d'insultes sera détectée par l'intersection de ces deux listes : une intersection égale à zéro validant un commentaire qui peut être publié.

1.2 Inconvénients du système actuel

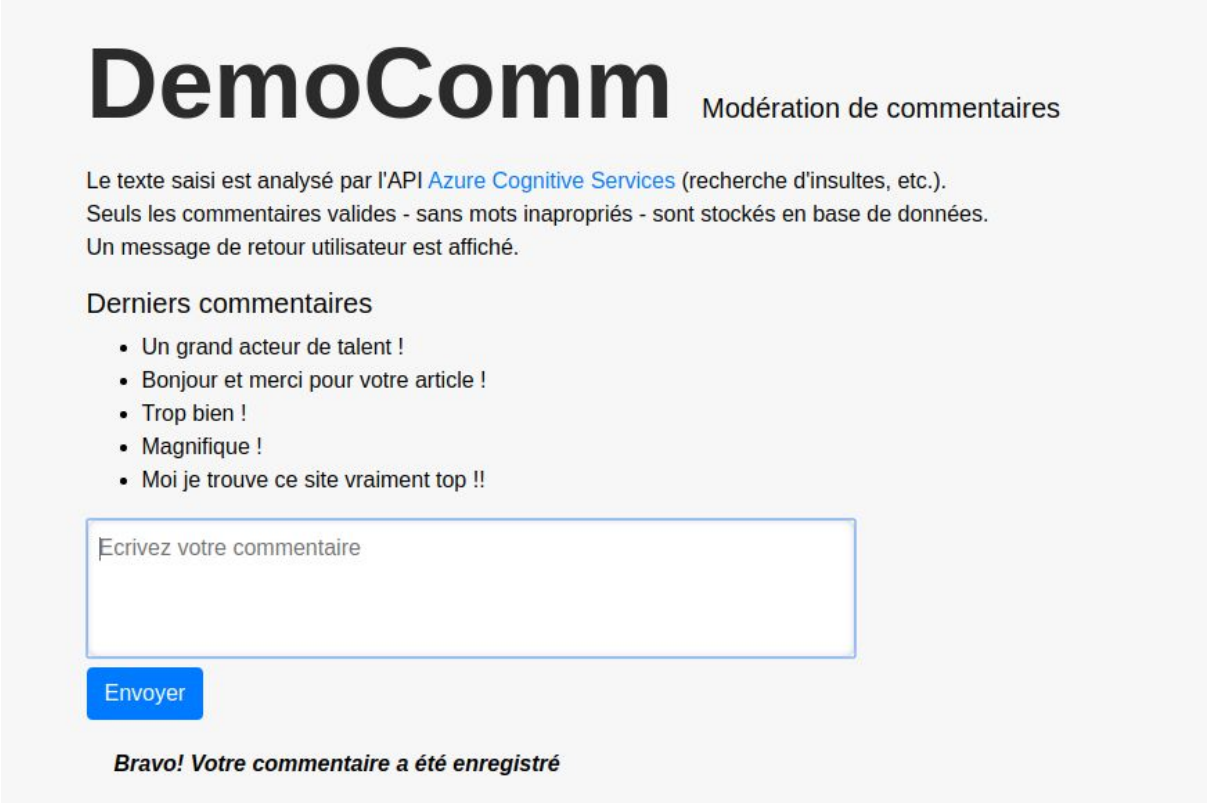
La gestion d'une liste de mots prohibés demande une intervention technique, par une mise à jour de manière volontaire - actualisation du dictionnaire d'insultes par exemple - et s'effectue indépendamment des commentaires reçus. Il n'y a pas de relation directe entre les commentaires précédents et les mots que l'on souhaite prohiber. Le système n'effectue aucun apprentissage.

La segmentation de commentaires - *tokenization* - mentionnée précédemment peut composer un lexique, mais il faut l'associer à une analyse syntaxique[5] pour mettre en évidence la structure d'un texte et obtenir les relations existantes entre les mots. Ce n'est pas le cas dans notre système. Et par opposition à ces phases, une analyse sémantique[6] permettrait d'établir les significations de groupes de mots en utilisant le sens des éléments du texte.

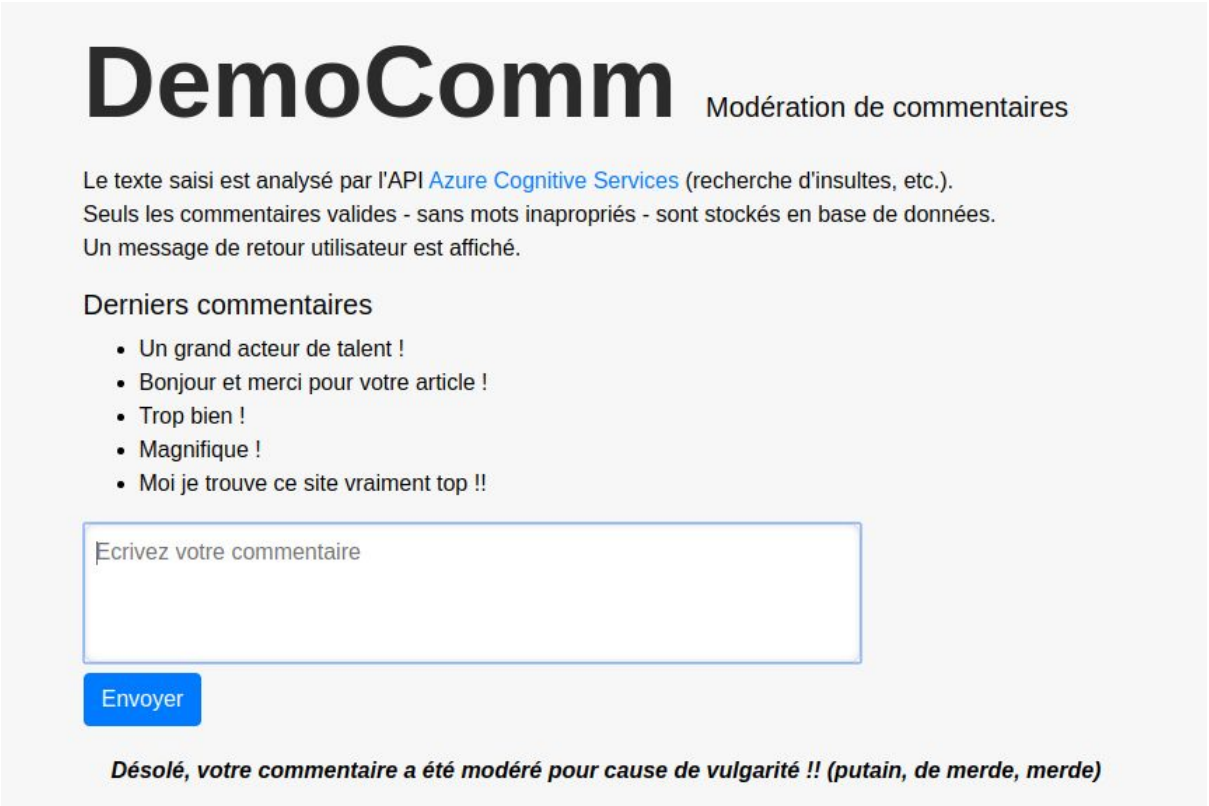
Le système actuel détecte uniquement les termes explicites (insultes et mots prohibés) et non les expressions implicites résultant de groupes de mots.

[4] https://fr.wikipedia.org/wiki/Analyse_lexicale
[5] https://fr.wikipedia.org/wiki/Analyse_syntaxique_de_la_langue_naturelle
[6] https://fr.wikipedia.org/wiki/Analyse_s%C3%A9mantique

1.3 Formulaire de saisie de commentaire (prototype)



Exemple de commentaire validé



Exemple de commentaire modéré

1.4 Fonction *listmoderation* (comparaison de listes)

```
@app.route('/comment/listmoderation', methods=['POST'])
def listmoderation():
    # Get comment from the POST body
    if request.method == "POST":
        req = request.form.to_dict()
        comment = req["comment"]
        comment_list = list(comment.split(" "))
        comment_list = [x.lower() for x in comment_list]

        # Recup forbidden words
        f = open("app_moderation/checklist.txt", "r")
        forbidden_list = f.read()
        forbidden_list = list(forbidden_list.split("\n"))
        forbidden_list = [x.lower() for x in forbidden_list]

        # lists compare intersection
        inter_list = list(set(comment_list) & set(forbidden_list))

        # Traitement commentaire
        if len(comment) != 0:
            if len(inter_list) == 0 :
                texte = "Bravo! Votre commentaire a été enregistré"
                # Enregistrement du commentaire en bd
                helper.add_to_list(comment)
            elif  len(inter_list) == 1:
                texte = "Votre commentaire a été modéré à cause du mot
{}".format(inter_list[0])
            elif len(inter_list) > 1:
                forbidden_words = ', '.join(inter_list)
                texte = "Désolé, votre commentaire a été modéré pour cause de vulgarité
!! ({})".format(forbidden_words)
        else:
            texte = "Aucun commentaire saisi."

        # Récup liste commentaires depuis bd
        res = helper.get_list()

        # Render template
        return render_template("index.html", message=texte, data=res)
```

Code Python de la fonction *listmoderation*

La fonction *listmoderation* est déclenchée par la route '/comment/listmoderation' lors de la soumission du formulaire, dont le contenu est récupéré grâce à l'objet *request*. Cette fonction effectue le traitement - tokenisation - et la comparaison de listes, puis retourne le résultat au template qui l'affiche à l'utilisateur.

2. L'état de l'art et les solutions existantes

2.1 Comparatif de solutions pour la modération automatisée

- **OVH Bodyguard**
  https://www.ovhcloud.com/fr/case-studies/bodyguard/
  Solution qui propose d'analyser le contexte dans lequel est formulé un commentaire, ainsi que de déterminer la ou les personnes à qui il s'adresse. Service managé, il offre la possibilité d'intégrer des algorithmes open source, tels que ceux proposés par scikit-learn.
- **Amazon Comprehend**
  https://aws.amazon.com/fr/comprehend/
  Service de traitement du langage naturel qui exploite l'apprentissage automatique pour identifier des informations et des relations dans un texte.
  - *Tarification* : Jusqu'à 10 millions d'unités → 0.0001 USD
- **Google Natural Language**
  https://cloud.google.com/natural-language?hl=fr
  Grâce au machine learning, l'API Natural Language révèle la structure et la signification d'un texte. Permet d'extraire des informations sur des personnes, des lieux et des événements, et mieux comprendre le sentiment général qui se dégage d'un contenu textuel.
  - *Tarification* : Gratuit de 0 à 5 000 unités envoyées puis 1$ pour 5001 à 1000000 d'unités.
- **Azure Content Moderator**
  https://docs.microsoft.com/fr-fr/azure/cognitive-services/content-moderator/overview
  Service cognitif qui vérifie le texte, les images et le contenu vidéo à la recherche d'éléments potentiellement dangereux, offensants ou indésirables.
  - *Tarification* : Gratuit pour 1 transaction par seconde puis pour 10 transactions par seconde → 0,844 € toutes les 1 000 transactions (Jusqu'à 1 million de transactions).

3. L'usage d'une IA plus évoluée avec Azure

Nous orientons le projet avec la solution Azure Content Moderator[7] car notre client utilise déjà plusieurs services Azure, et les fonctionnalités proposées correspondent aux traitements souhaités. Les éléments indésirables d'un texte sont détectés par l'API qui retourne les indices exploitables.

3.1 Fonction *moderation* : Traitement de résultat d'API

```
@app.route('/comment/moderation', methods=['POST'])
def moderation():
    # Get comment from the POST body
    if request.method == "POST":
        req = request.form.to_dict()
        comment = req["comment"]
        result_moderation = moderation_api(comment)

        # Traitement commentaire
        if len(comment) != 0:
            if result_moderation["Terms"] == None :
                texte = "Bravo! Votre commentaire a été enregistré"
                # Enregistrement du commentaire en bd
                helper.add_to_list(comment)
            elif  len(result_moderation["Terms"]) == 1:
                texte = "Votre commentaire a été modéré à cause du mot
{}".format(result_moderation["Terms"][0]['Term'])
            elif len(result_moderation["Terms"]) > 1:
                rejected_words = [item['Term'] for item in
result_moderation["Terms"]]
                rejected_words = ', '.join(rejected_words)
                texte = "Désolé, votre commentaire a été modéré pour cause de
vulgarité !! ({})".format(rejected_words)
        else:
            texte = "Aucun commentaire saisi."

        # Récup liste commentaires depuis bdS
        res = helper.get_list()

        # Render template
        return render_template("index.html", message=texte, data=res)
```

Code Python de la fonction *moderation*

[7] https://azure.microsoft.com/fr-fr/services/cognitive-services/content-moderator/

Le texte récupéré par le formulaire est passé en paramètre de la fonction *moderation_api* (ci-dessous), qui le soumet au service Azure Content Moderator. L'API retourne un objet json avec les mots à modérer, qui est traité en conséquence.

3.2 Fonction moderation_api : Appel à l'API *Azure Content Moderator*

```
import requests, urllib, json, config

# moderation api
def moderation_api(comment):
    # Request headers
    headers = {
        'Content-Type': 'text/plain',
        'Ocp-Apim-Subscription-Key': config.api_key
    }
    # Request parameters
    params = ({'classify': 'True'})
    body = [{'text' :comment}]
    url = config.api_url
    r = requests.post(url, json = body, params = params,headers= headers
)
    return r.json()
```

3.3 Remarques sur la version française d'Azure Content Moderator

Les fonctionnalités d'Azure Content Moderator pour la langue française sont moindres que celles proposées pour la langue anglaise: Le service ne réalise pas d'analyse de sentiments, et ne renvoie qu'une liste de mots prohibés traités de manière lexicale, sans analyse sémantique. Il traite les mots isolés et non pas le sens de groupes de mots.

Ces inconvénients sont communs à ceux observés avec la solution par comparaison de listes. Mais d'autres arguments plaident en faveur de la solution Azure Content Moderator : la gestion externalisée du service et ses mises à jour automatiques en particulier.

4. Evaluations & tests

Afin de garantir la continuité des fonctionnalités de notre application, nous réalisons une série de Tests de Non Régression en nous basant sur des tests unitaires à l'aide de pytest[8], puis un ensemble de tests fonctionnels par comparaison.

4.1 Tests unitaires sur les fonctions de modération avec pytest

```python
import pytest
import sys
import os
from main import app
from queries.moderation import moderation_api
import config

def test_hello():
    response = app.test_client().get('/')
    assert response.status_code == 200

def test_moderation():
    # Get comment from the POST body
    comment = 'Ceci est une phrase de test.'
    result_moderation = moderation_api(comment)
    assert isinstance(result_moderation, dict) == True

def test_listmoderation():

    # Get comment from the POST body
    comment = 'Ceci est une phrase de test.'
    comment_list = list(comment.split(" "))
    comment_list = [x.lower() for x in comment_list]
    assert isinstance(comment_list, list) == True

    # Recup forbidden words
    f = open("app_moderation/checklist.txt", "r")
    forbidden_list = f.read()
    forbidden_list = list(forbidden_list.split("\n"))
    forbidden_list = [x.lower() for x in forbidden_list]
    assert isinstance(forbidden_list, list) == True
```

Code Python de tests unitaires

[8] https://docs.pytest.org/en/latest/

4.2 Tests de Non Régression : Comparaison App1(listes) & App2 (API Azure)

ID Test	Description (mots testés)	Résultat app 1	Résultat app 2	Succès app 1	Succès app 2
1	Bonjour, je vous félicite pour votre site magnifique !	Bravo! Votre commentaire a été enregistré	Bravo! Votre commentaire a été enregistré	1	1
2	Bonjour je me permets de critiquer votre application de merde ...	Votre commentaire a été modéré à cause du mot merde	Désolé, votre commentaire a été modéré pour cause de vulgarité !! (de merde, merde)	1	1
3	Vos réflexions idiotes sur le Deep Learning me font un peu chier ...	Bravo! Votre commentaire a été enregistré	Votre commentaire a été modéré à cause du mot chier	0	1
4	Vos critiques idiotes sont un peu style trou du cul, non ?	Bravo! Votre commentaire a été enregistré	Votre commentaire a été modéré à cause du mot trou du cul	0	1
5	Ce mec est un vrai connard !	Votre commentaire a été modéré à cause du mot connard	Votre commentaire a été modéré à cause du mot connard	1	1
6	Putain de merde ! C'est du lourd !	Désolé, votre commentaire a été modéré pour cause de vulgarité !! (putain, merde)	Désolé, votre commentaire a été modéré pour cause de vulgarité !! (putain, de merde, merde)	1	1
7	Coucou ! Va te faire mettre ;)	Bravo! Votre commentaire a été enregistré	Bravo! Votre commentaire a été enregistré	0	0
9	Un peu casse couilles ce truc !	Votre commentaire a été modéré à cause du mot couilles	Votre commentaire a été modéré à cause du mot couilles	1	1
10	Un peu casse-couilles ce truc !	Bravo! Votre commentaire a été enregistré	Désolé, votre commentaire a été modéré pour cause de vulgarité !! (casse-couilles, couilles)	0	1
11	C'est qui la poufiasse qui a écrit ça?	Bravo! Votre commentaire a été enregistré	Votre commentaire a été modéré à cause du mot poufiasse	0	1
				Score App 1	Score App 2
				5	9

Conclusion

Les tests de Non Régression ont pour objectif de tester la continuité des fonctionnalités de notre application, et montrent que la solution faisant appel à l'API Azure Content Moderator fonctionne techniquement bien, avec un score de satisfaction supérieur à la solution précédente.

Les tests illustrent des différences de traitements et de subtilités entre les deux solutions. Aussi, l'un des grands intérêts d'Azure Content Moderator repose dans la maintenance automatique gérée par ce service, basé sur un modèle entraîné par Machine Learning.

Azure Content Moderator peut également vérifier la présence d'éventuelles informations d'identification personnelle dans les messages publiés. Ainsi, le service d'Azure propose des fonctionnalités intéressantes qui peuvent soit être appliquées, soit nous inspirer en tant que développeur. Par exemple, les meilleurs résultats de modération de contenu proviennent de la collaboration entre des opérateurs humains et des machines. Azure propose ainsi un outil de révision lorsque la confiance dans les prévisions peut être améliorée ou tempérée dans un contexte réel. Il revient aux concepteurs de projets de sélectionner entre les solutions commerciales disponibles ou bien customiser leurs propres solutions.

Informations annexes

- Code du projet
  https://github.com/flabastie/text_moderation