



Rapport sur l'optimisation d'un projet IA

Optimisation des hyperparamètres et monitoring du modèle

SARLAT Ludivine
07/06/2021

Table des matières

1.	Introduction.....	- 2 -
1.1	Cadre du projet :.....	- 2 -
1.2	Périmètre :.....	- 2 -
Objectif N°1	- 2 -	
Objectif N°2 :	- 2 -	
2	Projet initial	- 2 -
2.1	Collecte des données :.....	- 2 -
2.2	Mise en base de données :.....	- 2 -
2.3	Création du modèle d'IA :.....	- 3 -
2.3.1	Préparation des données - Preprocessing.....	- 3 -
2.3.2	Entrainement d'un modèle de vectorisation	- 4 -
2.3.3	Choix d'un modèle de classification	- 4 -
2.3.4	Tuning des hyperparamètres	- 5 -
2.4	Mise à disposition du modèle	- 6 -
2.4.1	Version POSTMAN.....	- 6 -
3	Amélioration fonctionnelle	- 7 -
3.1	Version Application Flask	- 7 -
3.2	Monitoring des requêtes API.....	- 8 -
3.3	Monitoring des performances du modèle en production.....	- 9 -
	Conclusion	- 10 -
	ANNEXES.....	- 11 -

1. Introduction

Je souhaite présenter le projet proposé initialement par notre formateur Simplon, Gilles Cornec, afin de mettre en évidence le travail de développement et d'amélioration d'un projet d'intelligence artificielle. Ce projet a pour but de fournir une application qui prédira le journal d'origine à partir d'un titre d'article.

1.1 Cadre du projet :

Les consignes initiales sont les suivantes: « *Monitored les performances d'un modèle de ML en production, (...) optimiser les hyperparamètres d'un modèle d'apprentissage afin d'obtenir la meilleure efficacité et mettre le meilleur modèle sous forme d'API. Dans un second temps, il est demandé de suivre les performances du modèle déployé via un dashboard sur lequel vous pourrez suivre ces performances en fonction de la métrique la plus pertinente selon votre problème* ».

1.2 Périmètre :

J'ai choisi de constituer mes données à partir de 4 journaux : *Le Monde*, *L'Express*, *Libération*, *Closer*. Si les 3 premiers sont relativement proches, l'hypothèse est que les thèmes abordés par le tabloïd sont suffisamment différents pour être plus facilement reconnus.

Objectif N°1 : Constituer un jeu de données qui permette de réaliser un modèle d'intelligence artificielle capable d'associer le titre d'un article à un journal qui aurait pu l'écrire.

Objectif N°2 :

Mettre en évidence le travail d'optimisation des modèles d'IA et réaliser le monitoring du modèle en production.

2 Projet initial

2.1 Collecte des données :

Les journaux choisis :

La collecte initiale s'est déroulée en décembre, janvier 2021 (renouvelée en mai 2021) et a permis de récupérer le titre de l'article, la date de parution, la catégorie associée (catégorie propre à chaque journal) pour les 4 journaux. Des Notebooks de récupération ont été créés pour chaque journal et propose en sortie un fichier csv contenant les données. Un Notebook de concaténation a permis d'agréger les données pour les 4 journaux sous format d'un fichier csv.

Le jeu de données initial contient 48 450 articles. La période de publication des articles est la même pour tous les journaux et va du 10-09-2020 au 15-05-2021. Les journaux ne sont toutefois de même pas représentés équitablement (respectivement 21 920 ; 12 092 ; 10 812 et 3626 pour *Le Monde*, *Libération*, *L'Express*, et *Closer*). En effet, *Le Monde* publie plus d'articles par jour que *Closer*.

Il avait été envisagé de prendre *Les Echos*, magazine plutôt en lien avec l'économie, mais les archives n'étaient disponibles que jusqu'en novembre 2018. Les données collectées auraient plus été biaisées par le décalage de collecte et donc par les termes abordés.

2.2 Mise en base de données :

La table principale a été créée par un script SQL et contient les colonnes suivantes (Fig 1):

- id_article : entier auto-incrémenté
- titre : chaîne de caractère

- jour_publication : date au format AAAA-MM-JJ
- categorie : chaîne de caractère
- journal : journal ayant publié l'article

Les données initiales ont été mises en base de données MySQL via un script python.

id_article	titre	jour_publication	categorie	journal
1	Pour les galeries d'art, une situation moins s...	2021-02-01	economie	Le Monde
2	Quand la masturbation provoque une hémorragie ...	2021-02-01	inconnue	Le Monde
3	Les pistes de la France pour améliorer les ter...	2021-02-01	economie	Le Monde
4	Facebook et Apple se livrent une guerre ouverte	2021-02-01	economie	Le Monde
5	Kent Walker : « Google agit selon les lois, et...	2021-02-01	pixels	Le Monde

Figure 1 : Dataframe relatif aux données de la table « articles » de la base « journaux ».

2.3 Création du modèle d'IA :

La problématique de ce projet est un problème classique d'IA de traitement du langage naturel (NLP) en vue d'une classification. On cherche à prédire une classe (un journal) à partir d'un titre (une chaîne de caractères). Nous avons utilisé la librairie SCIKIT-LEARN et des méthodes spécifiques au traitement du langage naturel (nltk, tfidf ...).

2.3.1 Préparation des données - Preprocessing

Cette étape consiste à préparer le jeu de données pour un modèle de classification de machine learning. En effet, les algorithmes d'apprentissage automatique s'attendent à des données au format matriciel où chaque ligne correspond à une observation et chaque colonne à une caractéristique ; et où les valeurs sont des valeurs numériques. Si les données contiennent des valeurs autres que numériques, il faut procéder à des encodages de ces valeurs dans un format numérique.

Les titres de journaux sont les éléments d'entrée au futur modèle et la sortie (la cible) est le journal d'origine (4 modalités). Les titres vont donc devoir être encodés : c'est-à-dire que chaque titre va être transformé en un vecteur à N colonnes (N= nombre de mots du corpus constitué avec l'ensemble des titres du jeu d'entraînement) et sera composé de 1 et de 0 selon que l'élément correspondant à la colonne sera présent ou non dans le titre.

Les titres de journaux ont été préparés de sorte à séparer tous mes mots, supprimer tous les termes inutiles ou vides de sens (= stop_words). Les méthodes « nltk » ont été utilisées notamment pour la récupération des « stop_words » français. D'autres mots ont été ajoutés afin de nettoyer le dataset (exemple : « c'est », « j'ai », « plus » , « contre »,etc.). Les mots ont été transformés en radicaux via la méthode « FrenchStemmer ».

Les mots issus du titre ont subi les traitements suivants :

- Mis en minuscules
- Suppression des caractères spéciaux
- Suppression des nombres
- Split des mots
- Radicalisation des mots (par exemple 'France', 'français', 'française' deviennent 'franc')

Ces différents traitements ont été choisis pour l'amélioration qu'il apportait au modèle (Fig 2).

La colonne « journal » est transformée en valeur numérique.

id_article	titre	jour_publication	categorie	journal
1	galer art situat moin sombr redout	2021-02-01	economie	1
2	quand masturb provoqu hémorrag méning	2021-02-01	inconnue	1
3	pist franc améior term accord libreéchang me...	2021-02-01	economie	1
4	facebook apple livrent guerr ouvert	2021-02-01	economie	1
5	kent walk googl agit selon lois non selon pol...	2021-02-01	pixels	1

Figure 2 : Dataframe des données suite au preprocessing.

2.3.2 Entrainement d'un modèle de vectorisation

Seules les colonnes « titre » et la colonne « journal » (cible) sont prises pour la création du modèle. Ces données ont été séparées en un jeu d'entraînement (= train) et un jeu de test (= test) (Figure 3).

```
1 x = df.titre
2 y = df['journal']
3 X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=0)
4 print(np.shape(X_train), np.shape(X_test), np.shape(y_train), np.shape(y_test))

(33915,) (14535,) (33915,) (14535,
```

Figure 3 : Code python pour séparer le jeu de données en train et test

Comme la proportion des différents journaux n'est pas la même sur l'ensemble du jeu de données, j'ai décidé d'appliquer une méthode de rééchantillonnage de la librairie IMBLEARN (RandomOverSample) afin que chaque journal apparaisse dans les mêmes proportions.

Ensuite, la méthode « `tfidfVectorizer` » a été appliquée afin de transformer l'ensemble des titres en une matrice numérique ayant autant de lignes que le dataframe initial (une ligne= un titre) et ayant autant de colonnes que de mots apparaissant dans le corpus de titres (ici 22618).

```
1 from sklearn.feature_extraction.text import TfidfVectorizer
2
3 tfidf = TfidfVectorizer(analyzer='word')
4 tfidf.fit(X_train)
```

Figure 4 : Code python pour l'entraînement de la méthode `tfidf`.

Une fois le modèle de transformation `tfidf` entraîné, il est appliqué sur les jeux de données (train et test). Ce modèle `tfidf` est sauvegardé au format « pickle » car il devra être à nouveau appliqué sur les nouvelles entrées qui appelleront le modèle d'IA.

2.3.3 Choix d'un modèle de classification

Plusieurs types d'algorithmes de classification de la librairie SCIKIT-LEARN ont été testés :

- Modèle KNN (KNeighborsClassifier)
- Modèle RandomForest (RandomForestClassifier)
- Modèle Support Vector Machines - SVM (SVC)
- Modèle Naive Bayes (MultinomialNB)

Afin de pouvoir comparer ces différents modèles, la librairie MLFLOW a été utilisée et permet d'enregistrer les différents « run » des modèles cités plus haut et de visualiser les paramètres et les performances des modèles via un dashboard (lancé depuis une ligne de commande via mlflow ui et qui appelle les fichiers du répertoire mlflow) (Fig 5).

Search Runs: <input "="" and="" tags.mlflow.source.type="LOCAL" tree"="" type="text" value="metrics.rmse < 1 and params.model = "/>										<input type="button" value="Filter"/>	<input type="button" value="Search"/>	<input type="button" value="Clear"/>
Showing 88 matching runs <input type="button" value="Compare"/> <input type="button" value="Delete"/> <input type="button" value="Download CSV"/>										<input type="button" value="Columns"/>		
							Parameters >			Metrics		
<input type="checkbox"/>	↓ Start Time	Run Name	User	Source	Version	Models	algorithm	best_params	cross_val	best_score	score	
<input type="checkbox"/>	⌚ 2021-05-20 18:08:13	-	utilisateur	📁 C\ProgramD\ -	-	sklearn	SVM	{'C': 1.0, 'bre...	[0.7995116 0.8029304 0....	-	0.589	
<input type="checkbox"/>	⌚ 2021-05-20 16:50:27	-	utilisateur	📁 C\ProgramD\ -	-	sklearn	RandomForest	{'bootstrap': ...	[0.80545381 0.81107041...	-	0.564	
<input type="checkbox"/>	⌚ 2021-05-20 16:36:33	-	utilisateur	📁 C\ProgramD\ -	-	sklearn	KNN	{'algorithm': ...	[0.62531543 0.63142043...	-	0.477	
<input type="checkbox"/>	⌚ 2021-05-19 15:27:49	-	utilisateur	📁 C\ProgramD\ -	-	sklearn	Naive_bayes	{'alpha': 1.0, '...	[0.62840303 0.62503508...	-	0.514	

Figure 5: Dashboard MLflow affichant les runs d'entraînements des modèles KNN, RandomForest, SVM et Naïve bayes.

Les performances modèles peuvent être évaluées selon plusieurs métriques : dans le cas présent, la métrique de référence sera la justesse de la prédiction(= **Accuracy**). Les métriques « Precision », « Recall » et « F1-score » sont aussi calculées mais n'apparaissent pas dans le dashboard. Le **modèle SVM semble être le modèle le plus performant avec une accuracy de 0,5896 (58%)**.

Une analyse plus détaillée grâce à la méthode « classification_report() » permet de voir qu'un modèle est performant pour prédire les articles du journal *Closer* (classe 4) mais l'est moins pour les autres journaux (exemple pour SVM : *Closer* à 0,87 et 0,60, 0,52 et 0,53 pour *Le Monde*, *Libération* et *L'Express*) (Fig 6).

```
: 1 print(classification_report(y_test, pred_svm))

precision    recall  f1-score   support

      1       0.60      0.76      0.67      6564
      2       0.52      0.40      0.45      3600
      3       0.53      0.42      0.47      3339
      4       0.87      0.73      0.80      1032

accuracy                           0.59      14535
macro avg       0.63      0.58      0.60      14535
weighted avg    0.58      0.59      0.58      14535
```

Figure 6: Résultats de la méthode classification_report() du modèle SVM entraîné

Ceci corroborerait notre hypothèse de départ qui était de dire que le tabloïd ne traite pas les mêmes sujets que les autres journaux ; et qu'il est donc plus facile pour le modèle de faire de bonnes prédictions sur ce journal.

2.3.4 Tuning des hyperparamètres

J'ai choisi de prendre le **modèle SVM** et d'appliquer les méthodes de tuning d'hyperparamètres afin d'améliorer les performances du modèle. Par défaut, le modèle SVM (méthode SVC) a les hyperparamètres suivants : C=1 ; kernel = 'rbf' ; degree=3 ; gamma='scale'.

La méthode « RandomizedSearchCV() » permet de balayer différents hyperparamètres et de choisir le meilleur set d'hyperparamètres : C = [0.01, 0.1, 1.0, 10, 50] ; kernel = ['poly', 'rbf', 'sigmoid']. J'ai choisi d'utiliser une méthode de cross-validation à 5 plis qui cherche le meilleur modèle en évitant le sur-apprentissage (ou overfitting).

La méthode randomizedSearch indique que le meilleur set d'hyperparamètres est C=50 ; kernel='rbf' ; gamma='scale'. L'accuracy du modèle optimisé est 0,6446 ; ce qui nous fait gagner 5%.

Le modèle optimum final serait donc un **modèle SVM (C=50, kernel=rbf ; gamma=scala)**.

Nous avons donc réalisé l'entraînement de ce modèle avec ces hyperparamètres et nous l'avons enregistré en format pickle afin qu'il soit téléchargeable par une application de mise à disposition.

					Parameters >		Metrics		
	Start Time	User	Source	VdModels	algorithm	best_params	cross_val	best_score	score
□	2021-05-13 22:30:37	Isarlat	□ C:\Users\Isarlat - sklean	SVM	↑ algorithm	{'C': 50, 'break_ties': False, 'cache_size': 20...}	[0.62312775 0.6246696 0.63053536 0.62392597 0.63207755]	-	0.646

Figure 7: Dashboard MLflow affichant les résultats du modèle SVM optimisé

2.4 Mise à disposition du modèle

Un moyen assez commun pour déployer le modèle est d'exposer des APIs REST. Grâce à l'architecture REST, il est possible de communiquer avec un service via un protocole http (comme pour consulter une page web) et d'invoquer des opérations disponibles (comme le calcul d'une prédiction).

2.4.1 Version POSTMAN

Dans le premier temps, nous avons créé un projet avec le framework FLASK qui permet de développer rapidement une application en langage Python. Ce projet est composé d'un fichier « api.py » qui contient les routes vers les urls interrogeables dans un navigateur. A l'initialisation du programme, le modèle d'IA et la méthode tfidf sont téléchargés au format pickle (préenregistrés lors de la partie de modélisation).

La route '/predict' permet de prendre en entrée un fichier au format json (contenant un titre potentiel d'article), de le transformer via la méthode tfidf et d'exécuter le modèle de prédiction pour finalement renvoyer la prédiction vers l'interface POSTMAN.

```
@app.route('/predict', methods=['POST'])
def predict():
    if lr:
        try:
            mon_article = request.get_json()
            article = nettoyage(mon_article[0]['titre'])
            query = tfidf_.transform([article])
            prediction = trouve_journal(lr.predict(query))
            return prediction

        except Exception as e:
            print("Une erreur s'est produite")
            return render_template("page.html")
        else:
            print ('Train the model first')
            return ('No model here to use')

    if __name__ == '__main__':
        try:
            port = int(sys.argv[1]) # This is for a command-line input
        except:
            port = 12345 # If you don't provide any port the port will be set to 12345

        with open("./src/models_pickle/file_model_fitted_svm_opti.pkl", 'rb') as file_model:
            lr = pickle.load(file_model)
            print ('Model loaded')

        with open("./src/models_pickle/file_tfidf.pkl", 'rb') as tfidf:
            tfidf_ = pickle.load(tfidf)
            print ('tfidf transfo loaded')

        app.run(port=12345, debug=True)
```

Le modèle de prédiction est d'abord mis à disposition via l'interface POSTMAN. Cette interface permet de choisir une méthode POST qui envoie un fichier json vers l'adresse <http://127.0.0.1:12345/predict>. En retour, l'API renvoie la prédiction, c'est-à-dire le titre du journal qui a pu l'avoir écrit.

The screenshot shows the POSTMAN interface with the following details:

- Request URL:** http://127.0.0.1:12345/predict
- Method:** POST
- Body Content:**

```
1 { "titre": "Une robe zéro déchet : merci patron !"}  
1 Libération
```
- Response Status:** Status: 200 OK
- Response Headers:** Time: 16 ms, Size: 164 B

Figure 8 : Interface de l'application POSTMAN qui envoie un titre de journal et reçoit la prédiction.

3 Amélioration fonctionnelle

3.1 Version Application Flask

Pour rendre l'accessibilité au modèle plus agréable, j'ai développé une interface en Flask qui est accessible via l'url « localhost:12345 ». Ce projet Flask contient les fichiers suivants (Fig 9) :

- Le fichier principal « api.py » contient :
 - la route '/' qui renvoie à la page d'accueil « page.html » ;
 - la route '/predict' qui est appelée lorsque le bouton de la page d'accueil est soumis. Lors de l'appel de cette route, le modèle est appelé, fait sa prédiction et renvoie vers la page « result.html » avec la prédiction du journal et les probabilités associées à chacun des journaux.
- Les templates (= pages graphiques html) :
 - page.html
 - result.html
- Le fichier « models.py » contient les fonctions pour nettoyer le titre proposé (=preprocessing) et pour interpréter le résultat du modèle (1=> *Le Monde* ; 2=> *Libération* ; ...).

```

> app
  \_ src
    \_ __pycache__
    \_ models_pickle
  \_ templates
    \_ page.html
    \_ result.html
  \_ api.py
  \_ models.py

```

```

<div class="hero-body">
  <div class="container">
    <form role="form" method="POST" action="/predict">
      <input type="text" name="monArticle" />
      <button type="submit" class="btn btn-default">Soumettre</button>
    </form>
  </div>
</div>

```

Figure 9 : Répertoire du projet Flask – Code python liant le bouton ‘soumettre’ de la page d’accueil à l’appel du service API « predict »..

L’interface d’accueil propose à l’utilisateur de donner le titre d’un journal et de soumettre ce titre en vue de faire une prédiction du potentiel journal source (Fig 10).

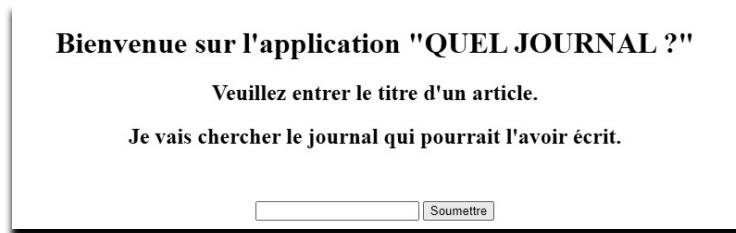


Figure 10 :Rendu visuel de la page d'accueil de l'application!

Le résultat de cette soumission est affiché sur une page « resultat.html » (Fig 11) , qui indique le nom du journal qui a le plus de chance de l’avoir publié. Les probabilités pour chacun des journaux s’affichent également. Cette fonctionnalité a nécessité de faire un entraînement du modèle SVM avec le paramètre « probability » = True. Par défaut, la méthode ne le fait pas.

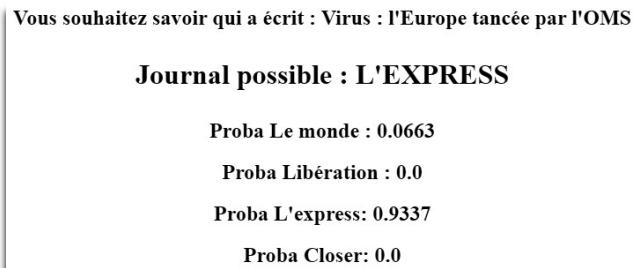


Figure 11 : Rendu visuel de la page indiquant la prédiction

3.2 Monitoring des requêtes API.

L’ajout du code suivant dans le fichier « api.py » permet de suivre les événements sur l’application via des logs enregistrés dans un fichier spécifique. Ces informations sur les requêtes API sont enregistrées mais ne sont pas exploitées actuellement. Cela pourrait être une piste d’amélioration.

```
logging.basicConfig(filename='app/logs/myapp.log', level=logging.DEBUG, format='%(asctime)s %(message)s', datefmt='%m/%d/%Y %I:%M:%S %p')
```

3.3 Monitoring des performances du modèle en production

Les performances du modèle déployé doivent rester bonnes dans le temps, autrement dit de nouvelles données (de nouveaux titres) appliquées au modèle doivent fournir un taux de classification tolérable. Si le taux est jugé trop bas, le modèle ne sera plus adapté et nous devrons redéployer un nouveau modèle (modèle existant ré-entraîné ou autre).

Pour ce faire, j'ai écrit un script python (donné en annexe) qui prend en entrée des titres de journaux (nouvelle période de référence) et teste les performances du modèle (accuracy) sur ces nouvelles données. Les méthodes mlflow sont utilisées dans ce script pour permettre la visualisation des performances du modèle sur le dashboard mlflow au cours du temps.

De plus, j'ai paramétré une fonction qui envoie un mail si les performances sont trop basses (Fig 13). La librairie smtplib permet de créer une alerte qui renvoie un mail à la maintenance (ici moi) via la messagerie Gmail. La figure 14 montre le type de mail reçu si les performances du modèle déployé sur les nouvelles données sont inférieures au seuil défini (ici 0.50).

Nous avons testé les performances du modèle avec de nouvelles données et nous les avons comparées à celles du jeu de données initial (utilisé pour la recherche du modèle optimal). Celles-ci diminuent d'un peu de plus de 10%. Elles passent de 0,6408 à 0,5699.

En complément de ce travail, nous pouvons aisément imaginer, un script indépendant (script de mise à jour), exécuté automatiquement via le planificateur de tâche de Windows, qui récupère mensuellement de nouvelles données. Celles-ci seraient utilisées par le script de monitoring afin de tester les performances du modèle déployé. Le script de monitoring alimenterait, mensuellement, le dashboard MLflow et nous alerterait si besoin.

```
email_text = """ The model performance is less than 0.50. Check it """
#email send request
try:
    server = smtplib.SMTP_SSL('smtp.gmail.com', 465)
    server.ehlo()
    server.login(gmail_user, gmail_password)
    server.sendmail(sent_from, to, email_text)
    print ('Email sent!')

except Exception as e:
    print(e)
    print ('Something went wrong...')
server.quit()
```

Figure 12: Code python du script de monitoring contenant le paramétrage pour l'alerting par mail.



Répondre | Répondre à tous | Transférer

Figure 13 : Mail reçu lorsque les performances du modèle déployé sont inférieures à 0.5

Conclusion

Ce travail m'a beaucoup plu car il m'a amenée à réaliser à nouveau les étapes classiques d'un projet d'IA : récupération de données (ici par du scrapping sur le net), nettoyage et préparation des données, mise en base données, choix d'un modèle d'IA, optimisation des hyperparamètres et déploiement de ce modèle via une interface Web et monitoring du modèle déployé.

Je pense qu'il me sera nécessaire d'améliorer ma pratique sur tous ces aspects au gré de mes futures expériences professionnelles. Il me tarde de mettre en action mes compétences liées au métier de développeur DATA IA pour de nouveaux projets.

ANNEXES

Tableau récapitulatif des métriques des modèles IA de classification

	KNN	RandomForest	SVM	Naïve Bayes
Accuracy	0.4746	0.5658	0.5896	0.5223
Precision	0.4787	0.5604	0.5843	0.5364
Recall	0.4774	0.5658	0.5896	0.5223
F1-score	0.4773	0.5596	0.5791	0.5204

Tableau récapitulatif des métriques du modèle en production avant et après une mise à jour de la base de données.

SVM opti	Dataset	Dataset 2
Accuracy	0.6408	0.5699
Precision	0.6455	0.5630
Recall	0.6408	0.5699
F1-score	0.6408	0.5504

Script de monitoring du modèle

```
import pandas as pd
import pymysql
from sklearn.metrics import precision_score , recall_score, f1_score , accuracy_score
import pickle
import mlflow.sklearn
from Flask_journaux.src.models import nettoyage, encode_journal
import smtplib

with open("Flask_journaux/src/models/pickle/file_model_fitted_svm_opti.pkl", 'rb') as file_model:
    lr = pickle.load(file_model)
    print ('Model loaded')

with open("Flask_journaux/src/models/pickle/file_tfidf.pkl", 'rb') as vector:
    tfidf = pickle.load(vector)
    print ('tfidf transfo loaded')

connection = pymysql.connect(host='localhost',
                                user='root',
                                password='XXXX',
                                db='journaux')

# create cursor
cursor = connection.cursor()
# chargement de nouvelles données
query_read = "SELECT * FROM articles where jour_publication<=date('2021-05-10') and jour_publication>=date('2020-04-21')"
cursor.execute(query_read)
raw_enregistrement = cursor.fetchall()
```

```

connection.commit()
df_test = pd.DataFrame(raw_enregistrement, columns =["id_article", "titre","jour_publication", "categorie","journal"])
cursor.close()
connection.close()

# preprocessing
X2 = df_test.titre.map(lambda x: nettoyage(x))
y2 = df_test.journal.map(lambda x: encode_journal(x))
X2_features = tfidf.transform(X2)

# calcul prediction avec le modèle chargé
pred2 = lr.predict(X2_features)
acc = accuracy_score(y2, pred2)
print('Accuracy', acc)
print('Precision', precision_score(y2, pred2 ,average='weighted'))
print('Recall', recall_score(y2, pred2,average='weighted'))
print('f1-score', f1_score(y2, pred2,average='weighted'))

with mlflow.start_run():
    print("Monitoring Model..")
    mlflow.sklearn.log_model(lr, 'SVM')
    mlflow.log_param('algorithm','SVM deployed')
    mlflow.log_param('best_params',lr.get_params(deep=True))
    mlflow.log_metric("score",acc)
    print("Fin de traitement")

if acc < 0.50:
    #email properties
    gmail_user = 'mariebolin@gmail.com'
    gmail_password = 'xxxx'
    sent_from = 'mariebolin@gmail.com'
    to = ["ludivinesarlat@gmail.com","ludivinesarlat@hotmail.com"]
    subject = 'Alert for model performance'
    email_text = """ The model performance is less than 0.70. Check it """
    #email send request
    try:
        server = smtplib.SMTP_SSL('smtp.gmail.com', 465)
        server.ehlo()
        server.login(gmail_user, gmail_password)
        server.sendmail(sent_from, to, email_text)
        print ('Email sent!')
    except Exception as e:
        print(e)
        print ('Something went wrong...')
    server.quit()

```