
ANALYSE DES SENTIMENTS DES DONNÉES NON-STANDARDS(TWITTER)

Résumé. —

Dans ce rapport, nous allons expliquer le déploiement d'un projet d'analyse des sentiments de tweets ce système est développé en utilisant **ALS** dans l'environnement **LogisticRegression**. Ce dernier existe dans une bibliothèque d'apprentissage automatique intitulé **scikit-learn** qui dispose d'une implémentation très efficace de **LogisticRegression** et finalement déployer notre modèle à l'aide **Docker**.

Nous allons détailler étape par étape dans le développement d'un modèle d'analyse de sentiment ensuite le déploiement en utilisant Flask et docker, Nous allons utiliser le jeu de données de tweets basé sur des évaluations négative ou positive.

Table des matières

1. Introduction.....	2
2. Exercice 1.....	2
3. Exercice 2.....	4
4. Exercice 3.....	4
5. Exercice 4.....	5
Documentation et sources.....	5

1. Introduction

L'analyse de sentiments est devenu une des approches les plus connus dans les réseaux sociaux soit pour limiter les messages de haines, raciste ... etc, dans ce rapport nous allons nous intéresser à un jeu de données provenant de twitter qui comporte deux notations soit négatif soit positif afin de pouvoir évaluer nos prochains tweets le but de cet algorithme est de pouvoir évaluer un tweet à partir de notre modèle déjà entraîné.

2. Exercice 1

Grâce à NLTK nous allons utiliser PorterStemmer afin de supprimer les affixes morphologiques des mots en ne laissant que le mot racine ensuite nous allons utiliser CountVectorizer pour transformer nos mots en vecteur, et juste avant nous allons transformer tout les labels neg ou pos en représentation binaire et finalement appliquer notre algorithme LogisticRegression la figure ci-dessous illustre les étapes détaillées.

```
data = pd.read_csv("sentiment.tsv", sep='\t')
# Features and Labels

data['label'] = data['label'].map({'pos':0, 'neg':1})
data['tidy_tweet'] = np.vectorize(remove_pattern)(data['tbody_ttexttt'], "@|\W|'")
tokenized_tweet = data['tidy_tweet'].apply(lambda x: x.split())

stemmer = PorterStemmer()
tokenized_tweet = tokenized_tweet.apply(lambda x: [stemmer.stem(i) for i in x])

for i in range(len(tokenized_tweet)):
    tokenized_tweet[i] = ' '.join(tokenized_tweet[i])

data['tidy_tweet'] = tokenized_tweet
data['body_len'] = data['tbody_ttexttt'].apply(lambda x: len(x) - x.count(" "))
data['punct%'] = data['tbody_ttexttt'].apply(lambda x: count_punct(x))
X = data['tidy_tweet']
y = data['label']

# Extract Feature With CountVectorizer
cv = CountVectorizer()
X = cv.fit_transform(X) # Fit the Data
X = pd.concat([data['body_len'], data['punct%'], pd.DataFrame(X.toarray())], axis = 1)

## Using Classifier
clf = LogisticRegression(C=0.1, class_weight=None, dual=False, fit_intercept=True,
                        intercept_scaling=1, l1_ratio=None, max_iter=100,
                        multi_class='auto', n_jobs=None, penalty='l2',
                        random_state=None, solver='liblinear', tol=0.0001, verbose=0,
                        warm_start=False)

clf.fit(X,y)
```

Après la création de notre algorithme nous allons maintenant pouvoir le déployer pour cela nous allons utiliser docker, pour faire cela nous allons commencer par créer notre fichier Dockerfile au préalable nous avons installé docker au par avant, ce fichier texte contient les commandes regrouper, À l'aide de docker

build, nous avons pu créer une build automatisée qui exécute successivement plusieurs instructions de ligne de commande.

```
FROM frolvlad/alpine-python-machinelearning:latest

RUN pip install --upgrade pip

WORKDIR /app

COPY . /app
RUN apk add build-base
RUN apk add --no-cache --virtual .build-deps g++ python3-dev libffi-dev openssl-dev && \
    apk add --no-cache --update python3 && \
    pip3 install --upgrade pip setuptools
RUN pip3 install -r requirements.txt
RUN python -m nltk.downloader punkt
EXPOSE 4000

ENTRYPOINT ["python"]

CMD ["app.py"]
```

La figure ci-dessus montre notre fichier docker avec le fichier app.py à exécuter et le installation nécessaire afin de satisfaire cela nous allons se baser sur le fichier requirements.txt qui contient les bibliothèques utilisés dont Flask qui est un framework Python qui facilite notre création d'une application web, notons que alpine est version légère de linux.

Après avoir utiliser docker build pour avoir notre image comme nous pouvons le voir dans ci-dessous :

```
root@ml:~# sudo docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
algo_container	v1	679bfacf97f9	14 seconds ago	620MB
twitter-sentiments	v1	1c571bf89838	18 hours ago	620MB
app	latest	60039793b31b	26 hours ago	414MB
mysql	latest	d4c3cafb11d5	34 hours ago	545MB
php	7-apache	2d5d57e31bd0	42 hours ago	414MB
gitlab/gitlab-ce	latest	ffadb2afd260	5 days ago	2.09GB
jenkins/jenkins	latest	e2036401c500	8 days ago	721MB
jupyter/all-spark-notebook	latest	5da9b337ac8c	3 weeks ago	4.11GB
docker/getting-started	latest	021a1b85e641	5 weeks ago	27.6MB

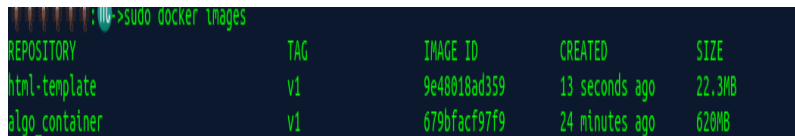
Maintenant nous pouvons utiliser la commande pour docker run comme ci-dessous :

```
root@ml:~# sudo docker run -p 4000:4000 -d --name algo algo_container:v1
2108dfd0a6e81633db1ad324fde62c5f68e793dffda9d2dfcb8f9403436a9697
root@ml:~# sudo docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
2108dfd0a6e8	algo_container:v1	"python app.py"	25 seconds ago	Up 24 seconds	0.0.0.0:4000->4000/tcp

3. Exercice 2

Suivant le même principe de l'exercice nous nous allons créer notre container dans le but de récupérer les résultats du container numéro 1 et les afficher voici notre container de templates qu'on utilise principalement du HTML simple et Flask :



```

$ sudo docker images
REPOSITORY          TAG         IMAGE ID      CREATED        SIZE
html-template        v1          9e48018ad359 13 seconds ago 22.3MB
algo_container       v1          679bfacf97f9 24 minutes ago 628MB
  
```

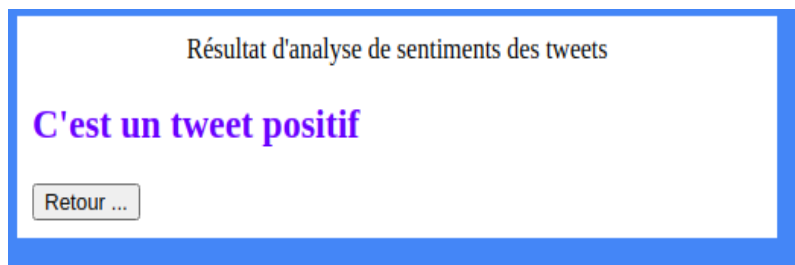
4. Exercice 3

Dans cette partie malheureusement nous n'avons pas pu utiliser Vagrant afin de résoudre cette problématique et répondre au but de l'exercice nous ne sommes basés sur un seul container afin d'afficher les résultats et voici ce que nous avons réussi à obtenir :



The screenshot shows a web application titled "Analyse de sentiments des tweets". It features a text input field with the placeholder text "Tester votre tweet ici :". The input field contains the text "Hello, it's a very good day !". Below the input field is a blue button labeled "Prédire".

La figure ci-dessus présente le champ pour pouvoir écrire notre tweets ou texte après nous passons à notre container dont il dispose d'un algorithme qui permet d'évaluer les sentiments ressentis par ce tweets et nous allons afficher le résultat comme le montre la figure ci-dessous :



The screenshot shows the result page of the web application. It has the title "Résultat d'analyse de sentiments des tweets". The main content is "C'est un tweet positif" displayed in large purple text. At the bottom left, there is a button labeled "Retour ...".

5. Exercice 4

Compose est un outil permettant de définir et d'exécuter des applications Docker multi-conteneurs, avec Compose, nous allons utiliser un fichier YAML pour configurer les services de notre application. Ensuite, avec une seule commande, créer et démarrer tous les services à partir de votre configuration. à l'aide de docker-compose up.

Documentation et sources

- [1] Towards data science
SENTIMENT ANALYSIS
<https://towardsdatascience.com/nlp-sentiment-analysis-for-beginners-e7897f976897>
 - [2] Youtube Video
EDUREKA
https://www.youtube.com/watch?v=sg1S7A532gM&t=2s&ab_channel=ThinkwithRiz
 - [3] Docker Hub
DOCKER
<https://hub.docker.com/>
-