



Rapport du projet NOSQL : Spam Detection avec Python et Cassandra



Cassandra



Projet de Gestion de Spam avec Python et Cassandra

Contexte et problématique

Le spam, ou courriel indésirable, est un problème majeur dans le domaine des communications numériques. Il encombre les boîtes de réception, réduit la productivité et peut souvent contenir des contenus malveillants tels que des liens de phishing ou des logiciels malveillants. Une solution automatisée et performante est essentielle pour détecter et filtrer ces messages en temps réel tout en assurant une gestion efficace des données à grande échelle.

Objectifs du projet

Ce projet vise à concevoir et implémenter un système de gestion de spam robuste, capable de :

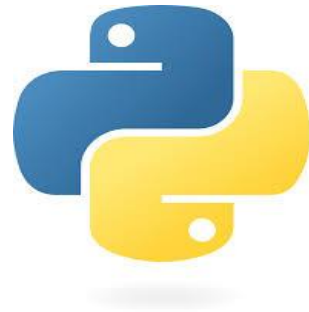
1. Identifier et classer les e-mails en **spam** ou **non-spam** avec un modèle d'apprentissage automatique.
2. Stocker, analyser et traiter efficacement de grandes quantités de données relatives aux e-mails grâce à une base de données distribuée.
3. Fournir une API permettant une prédiction en temps réel pour des applications externes.

Technologies utilisées

1. Python :

Python est le cœur du projet grâce à ses bibliothèques riches pour le traitement des données et l'apprentissage automatique :

- **NLTK et spaCy** pour le traitement du langage naturel (prétraitement des e-mails).
- **Transformers** (Hugging Face) pour l'entraînement et l'utilisation d'un modèle avancé comme **DistilBERT**.
- **Flask** pour développer une API web permettant des prédictions en temps réel.
- **Scikit-learn** pour des modèles de base (Naïve Bayes) en comparaison avec DistilBERT



2. Cassandra :

Cassandra est utilisée comme base de données NoSQL pour sa capacité à gérer de grandes quantités de données textuelles avec une haute disponibilité et des performances élevées :

- Stockage des messages d'e-mails et de leurs métadonnées (expéditeur, sujet, contenu, etc.).
- Conservation des prédictions du modèle pour un suivi et une analyse ultérieure.



- Gestion des données historiques pour entraîner et mettre à jour les modèles de machine learning.

3. Docker

Docker est une **plateforme open source** qui permet de **créer, déployer et exécuter des applications** dans des conteneurs. Un conteneur est une unité légère et portable qui regroupe une application et toutes ses dépendances (bibliothèques, fichiers de configuration, etc.), garantissant qu'elle fonctionne de manière cohérente dans n'importe quel environnement, qu'il s'agisse d'un ordinateur local, d'un serveur ou du cloud.

Docker repose sur des fonctionnalités de virtualisation du système d'exploitation (OS-level virtualization), ce qui le rend plus léger et plus rapide que les machines virtuelles classiques.



Architecture du projet

1. Collecte et stockage des données :

- Les e-mails (texte brut et métadonnées) sont stockés dans la base de données Cassandra.
- Les données peuvent être labellisées manuellement ou automatiquement (spam/non-spam).

2. Traitement et prétraitement des données :

- Les e-mails sont extraits depuis Cassandra et nettoyés (suppression des caractères spéciaux, conversion en minuscules, suppression des mots vides).
- Les textes sont préparés pour être vectorisés (pour les modèles traditionnels) ou tokenisés (pour les modèles avancés comme DistilBERT).

3. Entraînement du modèle :

- **Modèles de base** : Entraînement d'un modèle Naïf Bayes ou de régression logistique.
- **Modèle avancé** : Utilisation de **DistilBERT** pour un meilleur traitement du contexte des e-mails.
- Les données stockées dans Cassandra sont utilisées pour alimenter le modèle.

4. API de prédiction :

- Une API Flask est développée pour permettre aux applications clientes de soumettre un e-mail et de recevoir une prédiction en temps réel.
- Les résultats des prédictions (spam/non-spam et probabilités associées) sont également enregistrés dans Cassandra pour audit ou analyse future.

Fonctionnalités principales

1. Détection de spam en temps réel :

- Analyse des e-mails entrants et classification instantanée (spam ou non-spam).

2. Stockage structuré des e-mails :

- Les e-mails et leurs prédictions sont sauvegardés dans Cassandra pour des analyses historiques.
3. **Amélioration continue :**
- Possibilité de réentraîner les modèles en utilisant les données enrichies dans Cassandra.
4. **Gestion distribuée des données :**
- Cassandra assure une haute disponibilité et une faible latence, même avec de grands volumes de données.

Bénéfices du projet

- **Précision accrue** grâce à l'utilisation de modèles avancés comme DistilBERT.
- **Gestion efficace des données** via Cassandra, permettant une évolution avec l'augmentation des volumes d'e-mails.
- **Temps de réponse rapide** pour des prédictions en temps réel via une API web Python.
- **Évolutivité** pour gérer des volumes croissants d'e-mails et intégrer des fonctionnalités supplémentaires (comme la détection de contenu malveillant).

Réalisation du projet :

Nous avons utilisé Docker pour déployer Apache Cassandra dans notre projet afin de simplifier la gestion de l'environnement de développement. Docker nous a permis de créer un conteneur isolé contenant tous les composants nécessaires à Cassandra, ce qui a facilité son installation et sa configuration. Grâce à Docker, nous avons pu éviter les conflits avec d'autres logiciels installés sur la machine hôte et garantir que l'environnement de développement soit identique pour chaque membre de l'équipe.

1. Configuration de Docker pour Cassandra








- Télécharger l'image officielle de Cassandra

Docker propose une image officielle de Cassandra sur Docker Hub. Pour la télécharger on a exécuté la commande suivante :

```
docker pull cassandra:latest
```

-Lancer un conteneur Cassandra

```
docker run --name cassandra -d -p 9042:9042 cassandra
```

	Name	Container ID	Image	Port(s)	CPU (%)	Last start	Actions
	 cassandra	e8bed5dca259	cassandra	9042:9042 	2.32%	5 hours ago	  

2. Accéder à Cassandra dans le conteneur

En utilisant cqlsh (Cassandra Query Language Shell).

```
C:\Users\pc> docker exec -it cassandra cqlsh
Connected to Test Cluster at 127.0.0.1:9042
[cqlsh 6.2.0 | Cassandra 5.0.2 | CQL spec 3.4.7 | Native protocol v5]
Use HELP for help.
cqlsh> _
```

3. Intégration de Cassandra avec Python

Installer le driver Cassandra

```
pip install cassandra-driver
```

Connecter Python à Cassandra (testCassandra.py)

```
from cassandra.cluster import Cluster

# Connexion à Cassandra
cluster = Cluster(['127.0.0.1']) # Adresse IP du conteneur
session = cluster.connect()


# Utiliser ou créer un Keyspace (base de données)
session.execute("""
CREATE KEYSPACE IF NOT EXISTS gestionspam
WITH replication = {'class': 'SimpleStrategy', 'replication_factor': '1'}
""")
session.set_keyspace('gestionspam')

print("Connexion à Cassandra réussie !")
```

4. Gestion de Cassandra avec Docker Compose :

Si vous avez plusieurs conteneurs dans votre projet (par exemple, un pour Cassandra et un pour l'API Flask), vous pouvez utiliser **Docker Compose** pour simplifier la gestion.

-Créez un fichier docker-compose.yml :

 docker-compose.yml

```
version: '3.8' # Spécifie la version du fichier de configuration Docker Compose.

services: # Déclare les services qui composent l'application.
  cassandra: # Définition du service Cassandra.
    image: cassandra:5.0 # Utilise l'image Cassandra version 5.0 depuis Docker Hub.
    container_name: cassandra # Donne un nom explicite au conteneur Cassandra.
    ports:
      - "9042:9042" # Mappe le port 9042 du conteneur (Cassandra) au port 9042 de l'hôte.
```

-Lancer les services avec Docker Compose

```
docker-compose up
```

5- Importation des données dans Cassandra à partir d'un fichier CSV

Afin de traiter les données relatives aux messages pour la détection de spam, il était nécessaire de les importer dans la base de données Cassandra. Le fichier CSV contenant les données des messages a été utilisé pour cette opération. Chaque ligne du fichier CSV correspond à un message avec une étiquette (soit "ham", soit "spam") et son contenu.

Pour ce faire, nous avons d'abord créé une table messages dans le keyspace gestionspam. La table contient trois colonnes : un identifiant unique pour chaque message (id), l'étiquette associée au message (label), et le contenu du message lui-même (content). Ce modèle permet d'organiser les données de manière à faciliter leur consultation et leur traitement ultérieur.

```
from cassandra.cluster import Cluster
import csv
import uuid

# Connexion au cluster Cassandra
cluster = Cluster(['127.0.0.1'])
session = cluster.connect()
session.set_keyspace('gestionspam')

# Création de la table `messages`
session.execute("""
    CREATE TABLE IF NOT EXISTS messages (
        id UUID PRIMARY KEY,
        label TEXT,
        content TEXT
    )
""")
print("Table 'messages' créée avec succès.")
```

Ensuite, une fonction Python a été développée pour lire le fichier CSV ligne par ligne. Chaque message du fichier CSV a été inséré dans la table Cassandra avec un identifiant unique généré automatiquement en utilisant le module uuid de Python. Pour assurer une compatibilité avec les caractères spéciaux dans le fichier CSV, un encodage latin-1 a été utilisé lors de la lecture du fichier.

```
# Fonction pour lire le fichier CSV et insérer les données dans Cassandra
def insert_data_to_cassandra(file_path):
    # Utiliser un encodage compatible
    with open(file_path, mode='r', encoding='latin-1') as csvfile:
        reader = csv.reader(csvfile)
        next(reader) # Ignorer l'en-tête du fichier CSV si présent

        for row in reader:
            # Vérifier si les lignes sont valides
            if len(row) >= 2:
                message_id = uuid.uuid4() # Générer un UUID unique
                label = row[0].strip() # Première colonne : "ham" ou "spam"
                content = row[1].strip() # Deuxième colonne : contenu du message

                # Insérer les données dans la table `messages`
                session.execute("""
                    INSERT INTO messages (id, label, content) VALUES (%s, %s, %s)
                    """, (message_id, label, content))
                print(f"Données insérées : Label: {label}, Content: {content}")

# Lire et insérer les données depuis le fichier CSV
insert_data_to_cassandra('spam.csv')
```

```
cqlsh:gestionspam> select label,content from messages limit 10;
```

label	content
ham	HAPPY NEW YEAR MY NO.1 MAN
ham	Why is that, princess? I bet the brothas are all chasing you!
ham	Yeah we do totes. When u wanna?
ham	Its going good...no problem..but still need little experience to understand american customer voice...
ham	Tell where you reached
ham	I hate when she does this. She turns what should be a fun shopping trip into an annoying day of how everything would look in her house.
spam	100 dating service cal;l 09064012103 box334sk38ch
ham	DO NOT B LATE LOVE MUM
ham	On ma way to school. Can you pls send me ashley's number
spam	Congrats! 2 mobile 3G Videophones R yours. call 09063458130 now! videochat wid your mates, play java games, Dload polyPH music, noline rentl.

(10 rows)

5-Développement d'une application web de détection de spam utilisant Cassandra et un modèle d'apprentissage automatique

Importation des bibliothèques et modules nécessaires

Importe les bibliothèques et modules pour le traitement du langage naturel, la manipulation des données, la connexion à la base de données Cassandra et l'entraînement du modèle d'apprentissage automatique.

```

from flask import Flask, request, jsonify, render_template
import pandas as pd
import re
from cassandra.cluster import Cluster
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import nltk
import logging

```

- **Flask** : Un micro-framework Python utilisé pour créer des applications web. Ici, il est utilisé pour créer des routes (endpoints) web.
- **pandas** : Une bibliothèque utilisée pour manipuler et analyser des données sous forme de tableaux (DataFrame).
- **re** : Un module pour travailler avec des expressions régulières, utilisé ici pour nettoyer le texte.
- **Cassandra** : Un système de gestion de base de données NoSQL. `Cluster` est utilisé pour se connecter à la base de données Cassandra.
- **sklearn** : Utilisé pour la création et l'entraînement de modèles d'apprentissage automatique. `CountVectorizer` pour la vectorisation de texte et `MultinomialNB` pour le modèle Naive Bayes.
- **nltk** : Bibliothèque pour le traitement du langage naturel, utilisée ici pour le nettoyage du texte (supprimer les stop words et lemmatiser les mots).
- **logging** : Pour enregistrer les messages de log, utile pour déboguer et surveiller l'exécution du programme.

Téléchargement des ressources NLTK

Télécharge les ressources nécessaires pour le traitement du texte (listes de mots à ignorer).

```

# Download required NLTK data
nltk.download('stopwords')
nltk.download('wordnet')

```

Configuration du système de journalisation (logging)

Configure le système de logging pour suivre l'exécution du programme et enregistrer les messages d'information ou d'erreur.

```

# Configure logging
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
logger = logging.getLogger()

```

Initialisation du lemmatiseur, du set de stopwords et de l'application Flask

Le lemmatiseur est utilisé pour réduire les mots à leur forme de base (par exemple "running" devient "run"). Ici, nous utilisons le `WordNetLemmatizer` de NLTK, qui se base sur WordNet, une base de données lexicales de l'anglais.

stop_words est un ensemble (set) contenant des mots courants en anglais (comme "the", "is", "in", etc.) qui sont généralement retirés du texte lors du prétraitement, car ils n'apportent pas beaucoup d'information significative. Nous utilisons la liste des stopwords fournie par NLTK pour l'anglais.

__name__ est une variable spéciale en Python qui indique que ce fichier est le module principal (celui qui est exécuté).

```
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))
app = Flask(__name__)
```

Connexion à la base de données Cassandra

On établit une connexion à Cassandra et puis on sélectionne la base de données (gestionspam)

```
# 1. Connect to Cassandra
def connect_to_cassandra():
    try:
        cluster = Cluster(['127.0.0.1'])
        session = cluster.connect()
        session.set_keyspace('gestionspam')
        logger.info("Connected to Cassandra successfully.")
        return session
    except Exception as e:
        logger.error(f"Error connecting to Cassandra: {e}")
        exit()
```

Récupération des données depuis Cassandra

Récupère les données des emails (texte et label spam/non-spam) depuis la base de données Cassandra et les prépare sous forme de DataFrame pandas.

```
# 2. Fetch data from Cassandra
def fetch_data_from_cassandra(session):
    try:
        rows = session.execute("SELECT content, label FROM messages")
        data = [{'email_text': row.content, 'is_spam': 1 if row.label == 'spam' else 0} for row in rows]
        logger.info("Data fetched successfully from Cassandra.")
        return pd.DataFrame(data)
    except Exception as e:
        logger.error(f"Error fetching data: {e}")
        return pd.DataFrame()
```

Prétraitement des emails

Applique plusieurs étapes de nettoyage et de transformation aux emails (suppression des caractères spéciaux, conversion en minuscules, lemmatisation, et suppression des mots inutiles).

```
# Fonction de prétraitement des emails
def preprocess_email(email):
    # Supprimer les caractères non alphanumériques
    email = re.sub(r'\W', ' ', email)

    # Convertir le texte en minuscules
    email = email.lower()

    # Remplacer les espaces multiples par un seul espace
    email = re.sub(r'\s+', ' ', email)

    # Lemmatiser les mots et supprimer les stopwords
    email = ' '.join([lemmatizer.lemmatize(word) for word in email.split() if word not in stop_words])

    # Retourner l'email prétraité
    return email
```

Entraînement du modèle de détection de spam

Prétraite les emails, les vectorise (conversion en représentations numériques), et entraîne un modèle de machine learning (Naive Bayes) pour classer les emails comme spam ou non-spam.

Modèle Naive Bayes

Principe de base :

Le modèle repose sur le théorème de Bayes :

$$P(C|X) = \frac{P(X|C) \cdot P(C)}{P(X)}$$

où C est une classe (ex. : spam ou non-spam) et X est une observation (ex. : un email).

Naive Bayes suppose que les caractéristiques (X_1, X_2, \dots, X_n) sont indépendantes les unes des autres, d'où le terme "naïf". Bien que cette hypothèse soit rarement vraie, elle donne de bons résultats en pratique.

Dans ce cas, le modèle considère les fréquences des mots dans un texte. Il est particulièrement adapté pour des données textuelles vectorisées, comme les emails.

```
# Fonction pour entraîner le modèle de détection de spam
def train_spam_detector(data):
    logger.info("Entraînement du modèle de détection de spam...")

    # Prétraiter le texte des emails
    data['cleaned_text'] = data['email_text'].apply(preprocess_email)

    # Vectoriser le texte des emails en utilisant des bigrammes pour mieux capturer le contexte
    vectorizer = CountVectorizer(ngram_range=(1, 2))
    # Transformation du texte en vecteurs
    X = vectorizer.fit_transform(data['cleaned_text'])
    y = data['is_spam'] # Variable cible (spam ou non)

    # Entraîner le modèle Naive Bayes
    model = MultinomialNB()
    model.fit(X, y)

    logger.info("Entraînement du modèle terminé.")
    return model, vectorizer
```

-MultinomialNB() : Il s'agit d'une implémentation de l'algorithme Naive Bayes adapté pour les données discrètes, comme les comptes de mots dans des emails. Il est particulièrement utilisé dans les problèmes de classification de texte, comme la détection de spam. L'idée de base de Naive Bayes est que chaque caractéristique (ici, chaque mot ou bigramme) contribue indépendamment à la probabilité que l'email soit un spam ou non.

-model.fit(X, y) : Cette méthode entraîne le modèle en utilisant les données d'entrée X (les vecteurs représentant les emails) et les étiquettes y (qui indiquent si l'email est un spam ou non). Le modèle apprend les relations entre les caractéristiques (les mots/bigrams) et l'étiquette (spam/non-spam) pendant l'entraînement.

Routes Flask pour l'interface web

- **Route principale (/) :** Affiche la page d'accueil de l'application web.
- **Route de prédiction (/predict) :** Accepte un email via une requête HTTP et retourne si l'email est un spam ou non à l'aide du modèle préalablement entraîné.

```
# Flask route to render the HTML page
@app.route("/")
def index():
    return render_template("index.html")

# Flask route to predict if an email is spam
@app.route("/predict", methods=["POST"])
def predict():
    email_text = request.json.get("email", "")
    if not email_text:
        return jsonify({"error": "No email text provided"}), 400

    cleaned_email = preprocess_email(email_text)
    email_vector = vectorizer.transform([cleaned_email])
    prediction = model.predict(email_vector)
    is_spam = bool(prediction[0]) # Convert numpy value to Python boolean

    return jsonify({"is_spam": is_spam})
```

Exécution du programme principal

- Connecte à la base de données, récupère les données, entraîne le modèle et lance l'application Flask si les données sont présentes. Sinon, un message d'erreur est enregistré.

```
if __name__ == "__main__":  
    # Connect to Cassandra and fetch data  
    session = connect_to_cassandra()  
    data = fetch_data_from_cassandra(session)  
  
    if not data.empty:  
        # Train the spam detection model  
        model, vectorizer = train_spam_detector(data)  
        app.run(debug=True)  
    else:  
        logger.error("No data available in the database.")
```

Interface de Détection de Spam en Temps Réel

Cette interface web permet aux utilisateurs de tester la détection de spam d'un email en temps réel via une conversation interactive avec un assistant de détection de spam. L'utilisateur peut saisir le contenu d'un email, qui sera ensuite envoyé au backend pour analyse. Le système renvoie une réponse indiquant si l'email est un "Spam" ou "Non Spam".

Spam Detection Assistant

Hello do you want to meet tomorrow at 6 PM?

Not Spam

Congratulations you've won 100K

Spam

Send

Conclusion

Le projet de gestion de spam combine la puissance de Python pour le traitement des données et l'apprentissage automatique avec la scalabilité de Cassandra pour le stockage et la gestion des données. Ce système est conçu pour répondre aux besoins des entreprises et des utilisateurs qui souhaitent filtrer efficacement les e-mails indésirables tout en disposant d'une solution évolutive et performante.