Série TP : SPARK

I. Préparation de l'Environnement PySpark

1. Installer PySpark

Google Colab ne vient pas avec PySpark préinstallé, alors nous allons l'installer. Dans une cellule Colab, exécutez :

```
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
!wget -q https://archive.apache.org/dist/spark/spark-3.1.1/spark-3.1.1-bin-hadoop2.7.tgz
!tar xf spark-3.1.1-bin-hadoop2.7.tgz
!pip install -q findspark
```

2. Configurer les Variables d'Environnement

Configurez les variables d'environnement pour que Colab puisse utiliser Spark. Exécutez ce code :

```
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.1.1-bin-hadoop2.7"
```

3. Initialiser PySpark

Lancez PySpark en important et initialisant findspark :

```
import findspark
findspark.init()
```

II. Travaux pratiques :

Exercice 1 : Comptage des mots

Objectifs

- Écrire et exécuter un programme de comptage de mots (*word count*) en utilisant PySpark.
- Observer et analyser les résultats.

Simulation de données :

Pour simuler une source de données, créons un fichier texte simple :

```
text_data = """Hello world
Hello Spark
Hello Hadoop
Spark and Hadoop are big data frameworks
Spark is fast and efficient"""
with open("input.txt", "w") as file:
    file.write(text_data)
```

Solution :

Voici le code pour effectuer le comptage de mots en utilisant PySpark :

```
from pyspark import SparkContext
# Initialiser le contexte Spark
sc = SparkContext.getOrCreate()
# Charger le fichier texte
text_file = sc.textFile("input.txt")
# Processus de comptage de mots
counts = (text_file.flatMap(lambda line: line.split(" "))
```

```
            .map(lambda word: (word, 1))
            .reduceByKey(lambda a, b: a + b))
# Collecter et afficher les résultats
output = counts.collect()
for word, count in output:
    print(f"{word}: {count}")
```

Explication du Code

- **flatMap** : Sépare chaque ligne en mots.
- **map** : Associe chaque mot au nombre 1.
- **reduceByKey** : Additionne les valeurs associées aux mêmes mots (c.-à-d., les comptes).
- **collect** : Récupère les résultats sur le nœud maître pour les afficher.

Exercice 2 : Analyse de Logs de Serveur Web Apache avec PySpark

Les fichiers de logs de serveur web contiennent des informations sur toutes les requêtes HTTP faites au serveur, y compris la source (adresse IP), l'heure, le chemin demandé, le code de statut HTTP, et d'autres détails. Analyser ces logs permet d'améliorer la sécurité, de comprendre le comportement des utilisateurs, et d'optimiser les performances du serveur.

Exemple de ligne dans un fichier de log Apache :

127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif HTTP/1.0" 200 2326

- Charger et nettoyer un ensemble volumineux de logs d'accès d'un serveur web.
- Analyser les modèles de trafic : pages les plus visitées, périodes de pointe, adresses IP fréquentes, etc.
- Identifier les tentatives d'accès non autorisé (ex. requêtes suspectes) et anomalies dans le trafic.

Simulation de données :

```
# Créer un fichier de logs avec plusieurs lignes
with open("access_log.txt", "w") as log_file:
    logs = [
        '127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif HTTP/1.0" 200 2326',
        '127.0.0.1 - john [10/Oct/2000:14:56:22 -0700] "POST /login HTTP/1.0" 404 721',
        '192.168.1.1 - alice [10/Oct/2000:15:22:13 -0700] "GET /home HTTP/1.1" 200 5312',
        '10.0.0.1 - bob [10/Oct/2000:16:18:45 -0700] "GET /about HTTP/1.0" 500 1087',
        '127.0.0.1 - frank [10/Oct/2000:16:35:01 -0700] "GET /contact HTTP/1.1" 200 1982',
    ]
    # Écrire chaque log dans le fichier
    for log in logs:
        log_file.write(log + "\n")
```

Exercice 3 : Classification de Documents Non Structurés avec PySpark et Deep Learning

Les entreprises qui gèrent de grandes quantités de documents textuels (descriptions de produits, articles d'actualité, etc.) ont besoin de classer leurs documents en différentes catégories (technique, santé, divertissement, etc.). Dans ce TP, vous allez construire un modèle de deep learning pour classer automatiquement les documents en fonction de leur texte.

- Charger, nettoyer et prétraiter des données textuelles non structurées avec PySpark.
- Créer des embeddings de texte avec Word2Vec et entraîner un modèle de deep learning pour classer les documents.
- Utiliser un modèle RNN ou Transformer (BERT) pour une classification efficace des documents.

Simulation de données :

```
import pandas as pd
# Exemple de données simulées
data = {
   "doc_id": range(1, 11),
   "text": [
      "Artificial intelligence is transforming the tech industry.",
      "Exercise is beneficial for both body and mind.",
      "The new movie release has broken several box office records.",
      "Machine learning enhances predictive analytics in business.",
      "Healthy diets are key to maintaining good health.",
      "The football team won the championship last night.",
      "Cybersecurity is a major concern for companies worldwide.",
      "Research shows the impact of meditation on mental health.",
      "Latest gadget reviews for the tech-savvy consumer.",
      "The tennis match was intense and thrilling to watch."
   ],
   "category": ["tech", "health", "entertainment", "tech", "health", "sports", "tech", "health", "tech",
"sports"]
}

# Créer un DataFrame Pandas et sauvegarder en CSV
df = pd.DataFrame(data)
df.to_csv("documents.csv", index=False)
```