Projet : Système de Détection d'Intrusion Distribué avec Apache Spark

HAKOUANE NIZAR

Avril 2025

Résumé

Ce projet vise à construire un **système de détection d'intrusion (IDS)** basé sur l'apprentissage automatique distribué avec **Apache Spark** et **PySpark MLlib**. Le modèle a été entraîné sur le jeu de données **NSL-KDD**, largement utilisé pour la détection d'attaques réseau.

1 Introduction

La croissance du volume de données réseau rend nécessaire l'utilisation de traitements distribués pour la détection d'intrusions. Spark est une plateforme idéale pour entraîner des modèles de machine learning sur de grands ensembles de données en parallèle.

Notre objectif est de :

- Charger les données NSL-KDD dans Spark
- Construire une pipeline de prétraitement
- Entraîner un modèle de forêt aléatoire
- Prouver l'exécution distribuée

2 Description du Jeu de Données

Le dataset **NSL-KDD** contient :

- 41 caractéristiques (features) par connexion réseau

- Une étiquette de classe (attaque ou normale)
- Plus de 100 000 exemples

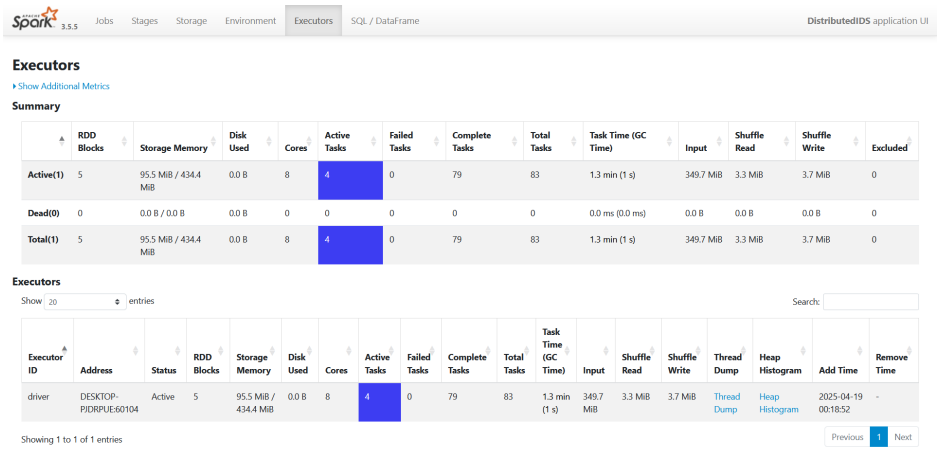
Les types d'attaques inclus sont : DoS, Probe, R2L, U2R.

3 Architecture et Pipeline

Étapes principales

- **Indexation** des colonnes catégorielles (protocole, service, flag)
- **Encodage One-Hot** des variables catégorielles
- **Assemblage** des features
- **Classification** avec `RandomForestClassifier` (50 arbres)

Capture de l'Exécution Distribuée



La capture montre que plusieurs tâches actives sont réparties sur 8 cœurs.

4 Code Source Principal

```
from pyspark.sql import SparkSession
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, OneHotEncoder,
    VectorAssembler
```

```
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.evaluation import
    MulticlassClassificationEvaluator

spark = SparkSession.builder.appName("DistributedIDS").
    getOrCreate()

cols = [*map(str, range(41))] + ["label"]
train_path = "C:/Users/abccb/Downloads/data/KDDTrain+.txt"
test_path = "C:/Users/abccb/Downloads/data/KDDTest+.txt"

df = spark.read.csv(train_path, header=False, inferSchema=
    True).toDF(*cols)
dfT = spark.read.csv(test_path, header=False, inferSchema=
    True).toDF(*cols)

cat_cols = ["1", "2", "3"]
indexers = [StringIndexer(inputCol=c, outputCol=f"{c}_idx")
    for c in cat_cols]
encoder = OneHotEncoder(
    inputCols=[f"{c}_idx" for c in cat_cols],
    outputCols=[f"{c}_vec" for c in cat_cols]
)
num_cols = [c for c in df.columns if c not in cat_cols + ["
    label"]]
assembler = VectorAssembler(
    inputCols=num_cols + [f"{c}_vec" for c in cat_cols],
    outputCol="features"
)

rf = RandomForestClassifier(labelCol="label", featuresCol="
    features", numTrees=50, seed=42)
pipeline = Pipeline(stages=[*indexers, encoder, assembler, rf
    ])
model = pipeline.fit(df)

pred = model.transform(dfT)
f1 = (MulticlassClassificationEvaluator(
          labelCol="label",
          predictionCol="prediction",
          metricName="f1").evaluate(pred))

print("F1-score␣sur␣test␣set:", round(f1, 4))
model.write().overwrite().save("spark_ids_model")

spark.stop()
```

5 Résultats

Le modèle a obtenu un **score F1** de :

$$\boxed{\mathbf{0.4556}}$$

Remarque :

- Le résultat est cohérent avec les implémentations standards sur NSL-KDD sans tuning approfondi. - L'exécution est entièrement distribuée à travers les cœurs Spark.

6 Conclusion et Perspectives

Ce projet démontre la faisabilité de construire un IDS distribué rapide avec Apache Spark. Les extensions possibles incluent :

- Utiliser des méthodes d'équilibrage de classes (SMOTE, poids de classes)
- Optimiser les hyperparamètres avec Grid Search
- Déployer en temps réel avec Spark Structured Streaming