A.D. 1308
unipg
UNIVERSITÀ DEGLI STUDI
DI PERUGIA

Tesina Finale di

Data Intensive Application and Big Data

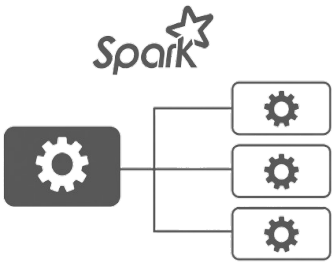
Corso di Laurea Magistrale in Ingegneria Informatica e Robotica - Data Science – A.A. 2024-2025

Dipartimento di Ingegneria

docente

Prof.Fabrizio Montecchiani

Distributed-vs-nonDistributed-ML



Studente

375437 **Emanuele Fossi** emanuele.fossi@studenti.unipg.it

0. Indice

1. Obiettivi del progetto

L'obiettivo del progetto è analizzare un dataset e addestrare un algoritmo di machine learning per effettuare una classificazione binaria. Verranno utilizzati diversi framework e strumenti per il calcolo parallelo, al fine di confrontarne le prestazioni.

1.1 Tecnologie utilizzate

Le tecnologie utilizzate nel progetto sono le seguenti:

- Google Cloud Platform
- Spark
- Python
- MLlib
- Dask
- Jupyter

1.2 Spark vs Dask

Apache Spark è un motore open-source per l'elaborazione distribuita dei dati, progettato per garantire elevate prestazioni e scalabilità. Consente l'analisi di grandi volumi di dati in parallelo su cluster di calcolo, supportando sia elaborazioni batch che in streaming. Grazie all'elaborazione in memoria (in-memory computing), Spark offre tempi di esecuzione ridotti e prestazioni ottimizzate. Fornisce API in diversi linguaggi, tra cui Scala, Python (tramite PySpark), Java e R, risultando così adatto a un'ampia gamma di applicazioni analitiche e di machine learning.

Dask, invece, è una libreria open-source nativa in Python, progettata per l'elaborazione parallela e distribuita. Estende strumenti noti come Pandas, NumPy e Scikit-learn, consentendo l'utilizzo di dataset di grandi dimensioni, anche superiori alla memoria disponibile, su sistemi multi-core o cluster. Grazie alla sua integrazione diretta con l'ecosistema scientifico Python, Dask facilita lo sviluppo di applicazioni scalabili in ambito analitico e computazionale. Sebbene entrambe le soluzioni siano pensate per l'elaborazione distribuita, differiscono per linguaggio, architettura e casi d'uso. Spark, sviluppato in Scala, è particolarmente adatto a contesti enterprise e a scenari in cui sono richieste elevate prestazioni su grandi cluster, inclusi flussi di dati in tempo reale. Dask, al contrario, si distingue per la semplicità d'uso in ambienti Python-based ed è ideale per flussi di lavoro flessibili, tipici del mondo scientifico e della ricerca, soprattutto su cluster di dimensioni contenute.

Entrambe le tecnologie sono state testate attraverso un cluster e delle VM costruite tramite **Google Cloud Platform (GCP)**. Google Cloud Platform (GCP) è un insieme di servizi cloud offerti da Google per supportare lo sviluppo, il deployment e la gestione di applicazioni e infrastrutture. Il servizio offerto da GCP non è gratuito; in ogni caso, Google mette a disposizione per i nuovi utenti 300€ di credito prima di iniziare a pagare i servizi che offre, dando la possibilità di provare molte delle prestazioni offerte.

1.3 Modello utilizzato

Il modello di machine learning scelto per il confronto è Random Forest che è un algoritmo di machine learning basato su un insieme di alberi decisionali, costruito mediante il processo di bagging (bootstrap aggregating). Ogni albero viene addestrato su un campione casuale dei dati di addestramento, mentre per la creazione di ciascun nodo viene selezionato casualmente un sottoinsieme di caratteristiche. Questo approccio riduce il rischio di overfitting e migliora la capacità di generalizzazione del modello. Durante la fase di previsione, i risultati degli alberi vengono aggregati, utilizzando il voto di maggioranza per compiti di classificazione e la media per la regressione. La Random Forest è ampiamente apprezzata per la sua robustezza, l'efficacia nella gestione di grandi volumi di dati e la sua capacità di mantenere buone prestazioni anche in presenza di variabilità nei dati.

2. Architetture Sviluppate e dataset

Per il progetto è stato implementato un cluster e una macchina virtuale, entrambi realizzati tramite i servizi di GCP.

2.1 Architetture Sviluppate

Il cluster è stato creato su Google Cloud Platform (GCP) tramite Dataproc, un servizio che consente di creare, gestire e scalare facilmente cluster per l'elaborazione di big data, supportando l'elaborazione batch, lo streaming e il machine learning. Il cluster in questione consiste in un nodo master con 2 vCPU e 4 GB di RAM, e nodi worker con specifiche analoghe. Dataproc gestisce automaticamente le connessioni tra i vari nodi del cluster, eliminando la necessità di configurare manualmente la rete. Inoltre, supporta nativamente Apache Spark, ma è necessario installare separatamente il notebook Jupyter.

Per quanto riguarda la VM implementata su GCP, questa deve essere configurata da zero. La creazione avviene tramite l'interfaccia web di GCP, mentre la configurazione si effettua collegandosi alla VM tramite SSH. All'interno della VM sono state quindi installate le versioni più recenti di Spark, Hadoop e Python. È naturalmente necessario installare anche Java, facendo attenzione alla versione scelta per evitare eventuali incompatibilità con le altre tecnologie. Per questo motivo, si è optato per l'installazione di Java 8. Dopo aver configurato la macchina virtuale, è possibile creare un ambiente virtuale dove installare Jupyter Notebook. Successivamente, per avviare il servizio, si esegue il comando:

```
1 jupyter notebook --no-browser --port=8888
```

Questo impedisce a Jupyter di aprire una nuova scheda nel browser e configura il server Jupyter per ascoltare sulla porta 8888. Poiché non è possibile accedere direttamente al browser tramite GCP, per interagire con l'interfaccia web di Ju-

pyter è necessario creare un tunnel SSH. Questo si realizza installando il servizio GCS sul proprio dispositivo e utilizzando il seguente comando:

```
gcloud compute ssh <nome-istanza> --project=<nome-progetto>
   --zone=<zona> -- -L 8888:localhost:8888
```

Questo comando consente di creare un tunnel SSH, permettendo di accedere comodamente all'interfaccia web di Jupyter sul proprio dispositivo.

2.2 Dataset

Il dataset utilizzato è disponibile al link `https://www.kaggle.com/datasets/hopesb/student-depression-dataset`. Questo contiene dati volti ad analizzare, comprendere e prevedere i livelli di depressione tra gli studenti. Alcune delle caratteristiche incluse nel dataset riguardano informazioni demografiche, performance accademiche, storia clinica relativa alla salute mentale, abitudini di vita e risposte a scale standardizzate di valutazione della depressione. La fase di preprocessing del dataset è stata effettuata separatamente rispetto ai codici per il calcolo distribuito, in modo da non intaccare i risultati ottenuti.

3. Dati Sperimentali

Per gli esperimenti sono quindi stati creati quattro script:

- **Script Spark su singola VM**: sviluppo su GCP, utilizzando una singola VM composta da 4 vCPU e 16GB di RAM, con caricamento dei dati tramite HDFS.
- **Script Spark su cluster**: sviluppo su GCP, utilizzando un cluster composto da 3 VM: 1 master e 2 worker, ciascuna con 2 vCPU e 4GB di RAM.
- **Script Dask su singola VM**: sviluppo su GCP, utilizzando una singola VM composta da 4 vCPU e 16GB di RAM, utilizzando Dask per la parallelizzazione insieme a scikit-learn.
- **Script non distribuito**: sviluppo locale su una macchina.

Grazie a questi script è stato possibile confrontare i vari tempi di addestramento dello stesso algoritmo utilizzando metodi distribuiti e non, su cluster reali oppure su macchine virtuali singole che li emulano.

I risultati ottenuti per i tempi di addestramento sono i seguenti:

- **Script Spark su singola VM**: 47.09 secondi
- **Script Spark su cluster**: 31.85 secondi
- **Script Dask su singola VM**: 3.86 secondi
- **Script non distribuito**: 2.35 secondi

4. Conclusioni

Essendo il dataset utilizzato di dimensioni contenute, i tempi di addestramento dell'algoritmo Random Forest risultano molto bassi. Tuttavia, da questi risultati possiamo comunque trarre alcune conclusioni.

Contrariamente alle aspettative, gli script che utilizzano il calcolo distribuito hanno impiegato più tempo rispetto allo script implementato in maniera classica, senza sfruttare il calcolo distribuito.

Questo comportamento può essere spiegato dal fatto che, per dataset di piccole dimensioni (in questo caso circa 1,4 MB), la vera potenza del calcolo distribuito non emerge. Strumenti come Spark e Dask sono progettati per gestire dataset dell'ordine dei terabyte.

In questo scenario, invece, scikit-learn è ottimizzato per dataset più piccoli e quindi riesce a performare meglio, mentre Spark e Dask introducono un overhead aggiuntivo nella gestione della parallelizzazione, che si traduce in tempi di esecuzione più lunghi.

Questi risultati evidenziano l'importanza di scegliere l'architettura computazionale più adatta in funzione della scala del problema, valutando caso per caso l'effettivo vantaggio del calcolo distribuito.