UNIVERSITA' DEGLI STUDI DI PERUGIA

Dipartimento di Ingegneria

Corso di laurea magistrale in Ingegneria Informatica e Robotica

curriculum Data-Science

Tesina di Big Data Management

Docente Fabrizio Montecchiani

Anno accademico 2022-2023

BENCHMARKS TRA FRAMEWORK SPARK E RAY CON MODELLO XGBOOST SU CLUSTER GCP

Tommaso Martinelli *350400*

Riccardo Rossi *346626*

1.Introduzione

Lo scopo del progetto è quello di fare predizione sul dataset di Kaggle, 'Rain in Australia' (https://www.kaggle.com/datasets/jsphyg/weather-dataset-rattle-package), utilizzando rispettivamente i due noti framework di calcolo, Ray e Spark, andandoli a testare in un ambiente distribuito come Google Cloud Platform al fine di analizzarne le prestazioni ed i risultati migliori.

2. DataFlow e tecnologie utilizzate

2.1 Il Dataset

Il Dataset 'Rain in Australia', scaricato da Kaggle, è un dataset che contiene circa 10 anni di osservazioni meteorologiche giornaliere registrate da molte località dell'Australia. Le 23 features, infatti, raccontano le caratteristiche metereologiche di una tipica giornata australiana: percentuale di umidità, direzione del vento, velocità del vento, temperatura giornaliera in diversi istanti della giornata, ecc…

Lo scopo del dataset è quello di fare classificazione binaria sulla label 'Rain_Tomorrow', ovvero riuscire a prevedere se, nella giornata successiva a quella per la quale si sono raccolte le features, pioverà oppure no, dunque RainTomorrow=1 (pioverà), oppure RainTomorrow=0 (non pioverà).

2.2 Tecnologie utilizzate

Le tecnologie utilizzate nel progetto sono:

- Jupyter notebook
- PyCharm
- Python
- Spark
- MLlib
- Ray

- GCP (Google Cloud Platform)

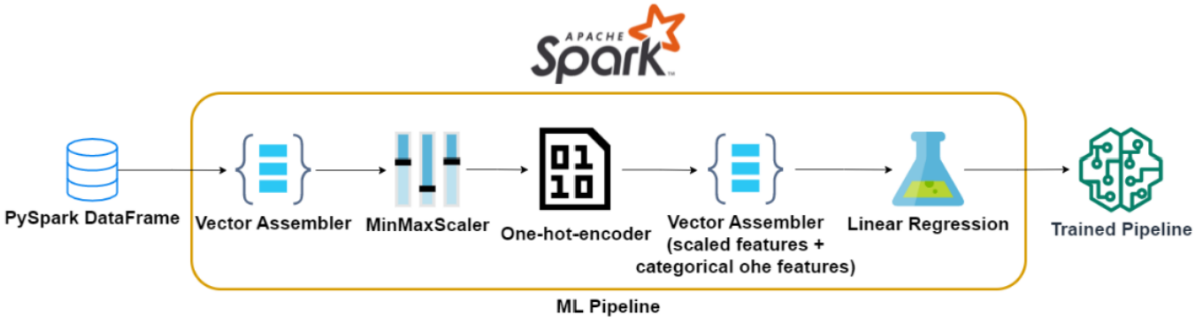
Una volta scaricato il dataset, si è deciso di fare EDA (Exploratory Data Analisys) separatamente dal resto del codice dedicato al calcolo distribuito, per una maggiore chiarezza di quest'ultimo. Una volta ottenuti i nuovi dati grazie all'EDA, è stato esportato il nuovo dataset ed è stato caricato quest'ultimo sia sullo script di Ray sia in quello di Spark.

Ray è un framework di calcolo unificato open source che semplifica la scalabilità dei carichi di lavoro di intelligenza artificiale. Affronta queste sfide a testa alta consentendo agli ingegneri e agli sviluppatori di machine learning di scalare facilmente i propri carichi di lavoro dai laptop al cloud senza la necessità di creare complesse infrastrutture di calcolo. Ray include Ray AI Runtime (**AIR**), un set nativo di librerie ML scalabili best-in-class, come ad esempio:

- ML data pre-processing tasks via *Ray Data*
- Training large models via *Ray Train*
- Hyperparameter tuning via *Ray Tune*

Inoltre, Ray e le sue librerie si integrano perfettamente con il resto dell'ecosistema Python e ML. Con queste librerie, i non esperti possono facilmente sfruttare il calcolo distribuito utilizzando i loro strumenti ML e Python preferiti. Nel progetto sono state riportate tutte e tre le librerie sopra citate, a eccezione della prima, il cui utilizzo è stato ridotto a seguito della scelta di fare EDA separatamente tramite PyCharm.

Apache Spark è un framework di elaborazione dati in grado di eseguire rapidamente attività di elaborazione su set di dati molto grandi e può anche distribuire attività di elaborazione dati su più computer, da solo o in tandem con altri strumenti di calcolo distribuito. La libreria di Spark dedita al machine learning che si è utilizzata nel progetto è MLlib. Quest'ultima standardizza le API per gli algoritmi di machine learning per semplificare la combinazione di più algoritmi in un'unica pipeline o flusso di lavoro:



Nel progetto sono stati riportati solamente i seguenti elementi della pipeline:

- Assembler
- Scaler
- Model

Questo perché gli altri strumenti servivano per la gestione delle variabili categoriche, le quali sono già state gestite diversamente come già ribadito più volte nel documento.

Il modello che abbiamo utilizzato per addestsrare i dati è XGBRF (Extreme Gradient Boosting Random Forest). Per capire come funziona bisogna prima introdurre alcuni concetti. Possiamo dire che l'effetivo modello utilizzato è un modello basato su alberi deciosionali del tipo random forest (RF), mentre con XGB che sta per Extreme Gradient Boosting, vogliamo sottolineare che a tale modello è stato applicato l'algoritmo del Gradient Boosting per la ricerca dei parametri ottimali e quindi minimizzare una funzione di perdita. Un breve cenno di come funziona:

Il termine "potenziamento del gradiente" deriva dall'idea di "potenziare" o migliorare un singolo modello debole combinandolo con una serie di altri modelli deboli al fine di generare un modello collettivamente forte. Tali modelli sono addestrati invece con l'algoritmo del gradient descent.

2.3 Architettura e DataFlow

Sono stati realizzati in tutto cinque script, i primi due, riguardanti la parte in Ray, gli ultimi tre inerenti a Spark. Ogni script rappresenta una diversa configurazione con la quale sono stati sviluppati quest'ultimi. Si è voluta implementare scalabilità orizzontale per Spark, aggiungendo così più nodi al cluster, mentre si è utilizzato un approccio verticale per Ray, andando a costruire una macchina molto potente in termini di calcolo:

1. ***Script Ray***: Sviluppo in locale su VM, con caricamento dati effettuato tramite HDFS.
2. ***Script Ray***: Sviluppo su GCP, con una VM composta da 16 vCPU e 60 GB di RAM.

1. ***Script Spark***: Sviluppo in locale tramite VM, con un caricamento dati effettuato tramite HDFS.

2. ***Script Spark***: Sviluppo su GCP, con un cluster composto da1 master e 3 worker, ciascuno dei quali avente 2 vCPU e 7,5 GB di RAM.
3. ***Script Spark***: Sviluppo su GCP, con un cluster composto da1 master e 3 worker, ciascuno dei quali avente 4 vCPU e 15 GB di RAM.

3. Casi d'uso

Per quanto concerne gli script eseguiti in locale, essi possono essere testati semplicemente andando a caricare il dataset nella propria VM, tramite HDFS. Una volta avviato Jupyter, possono essere caricati gli script (.pynb) accessibili attreverso il seguente link di github: https://github.com/tommasomartinelli/BigData. Per quanto riguarda Gli script eseguiti in GCP, per prima cosa si necessita di un account GCP, una volta creato può essere impostato un cluster secondo le configurazioni presenti nel file di README raggiungibile sempre tramite il link citato in precedenza e di conseguenza caricare gli script presenti nella medesima cartella di github.

4. Risultati

Partendo da Ray, possiamo notare che:

1. La prima configurazione ha impiegato 38.94 s.
2. La seconda configurazione ha impiegato 16.601 s.

Per quanto riguarda Spark:

1. La prima configurazione ha impiegato 33,935 s.
2. La seconda configurazione ha impiegato 44,898 s.
3. La terza configurazione ha impiegato 35.537 s.

Possiamo notare che nella configurazione ***2*** di Ray, i tempi si riducono di più della metà rispetto alla configurazione ***1***. Inoltre, è stata implementata anche una versione di Ray con model selection per la scelta degli iperparametri ottimi, ottenendo un tempo di esecuzione pari a 701 s per la

versione in GCP e 1151 s per la versione in locale. Mentre per quanto riguarda Spark è consigliabile l'implementazione della configurazione ***1***, questo perché non si hanno ritardi dovuti alla comunicazione in rete, inoltre se il dataset non è eccessivamente grande, il caricamento tramite HDFS permette tempi di caricamento più brevi. Infine, va anche notato che le macchine adibite al calcolo degli script in GCP si trovano in una posizione geografica lontana dalla zona in cui ci si trova, questo dovuto al fatto che era l'alternativa più vantaggiosa in termini economici. In generale in un applicazione non troppo dispendiosa in termini computazionali, anche l'utilizzo di una singola macchina può dare buoni risultati. Ray risulta essere una buona alternativa a Spark per applicazioni di questo tipo.

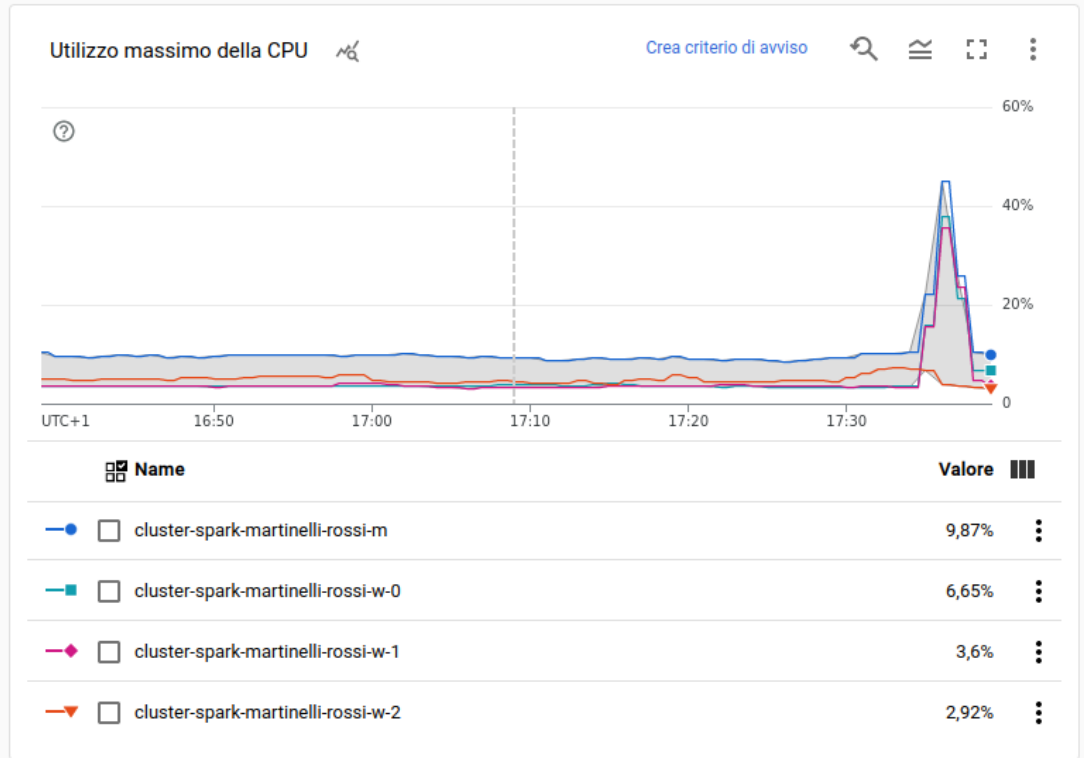


Fig1: utilizzo massimo CPU, Spark, configurazione 2.

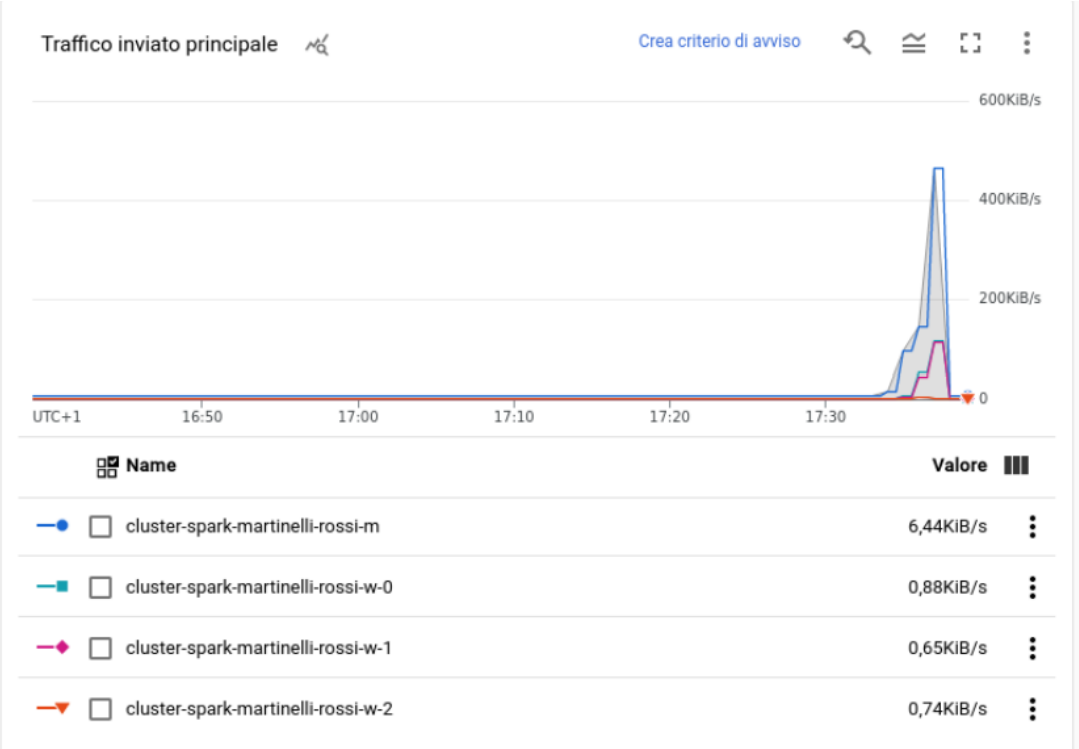


Fig2: traffico inviato, Spark, configurazione 2.

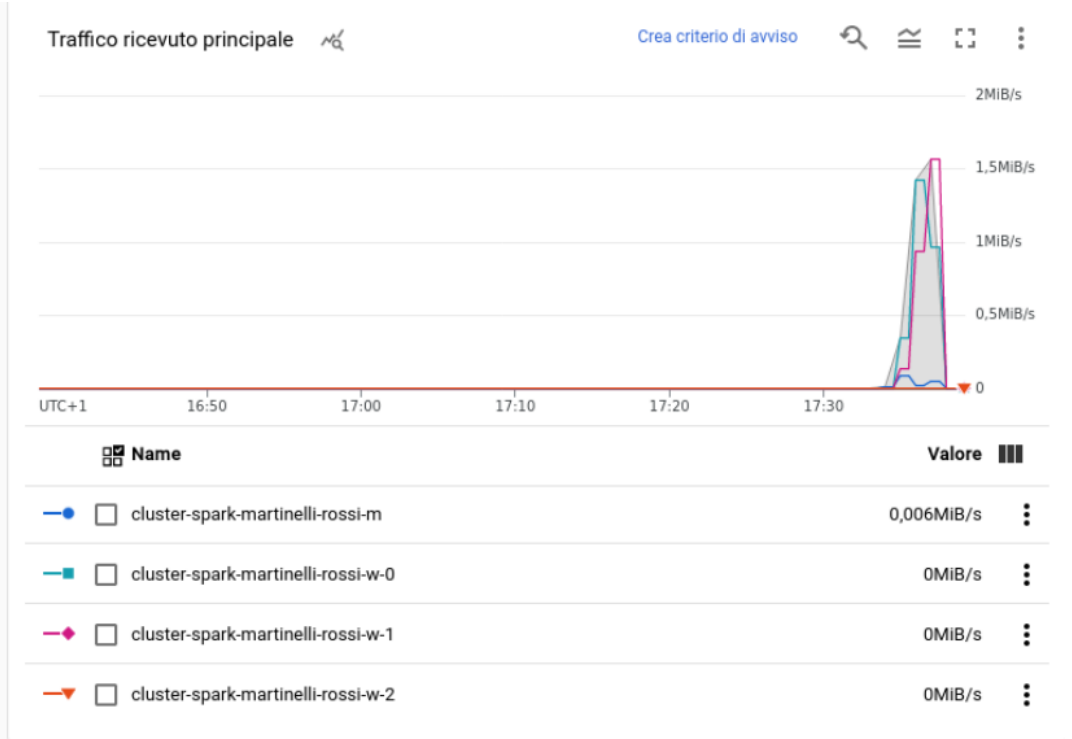


Fig3: traffico ricevuto, Spark, configurazione 2.

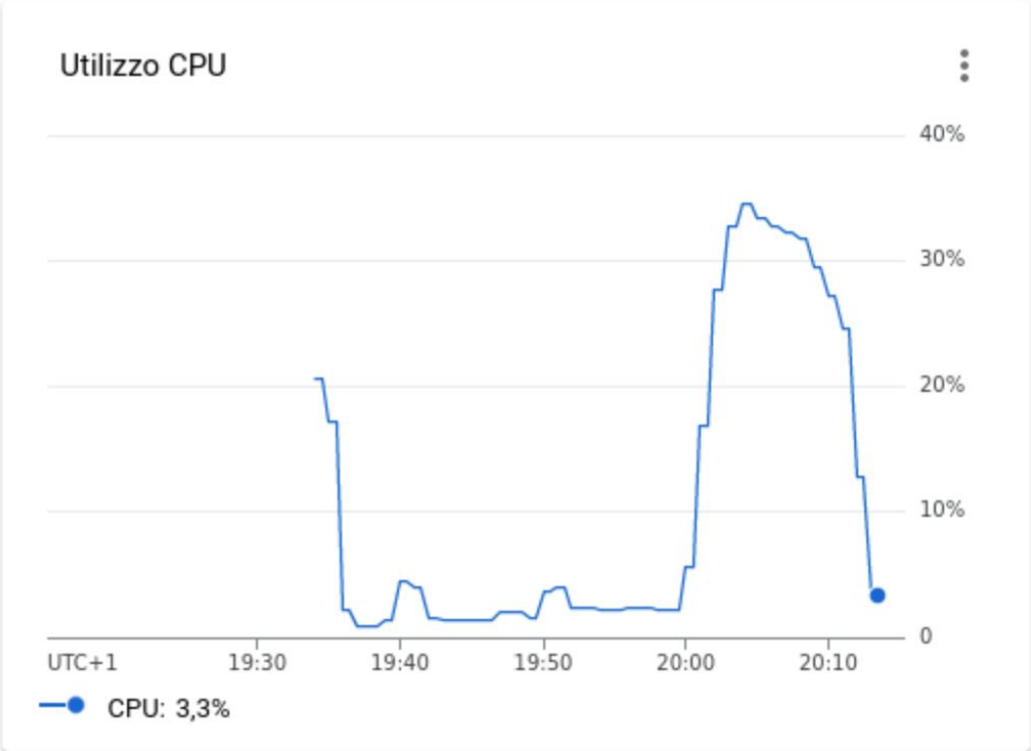


Fig4: utilizzo CPU, Ray, configurazione 2.

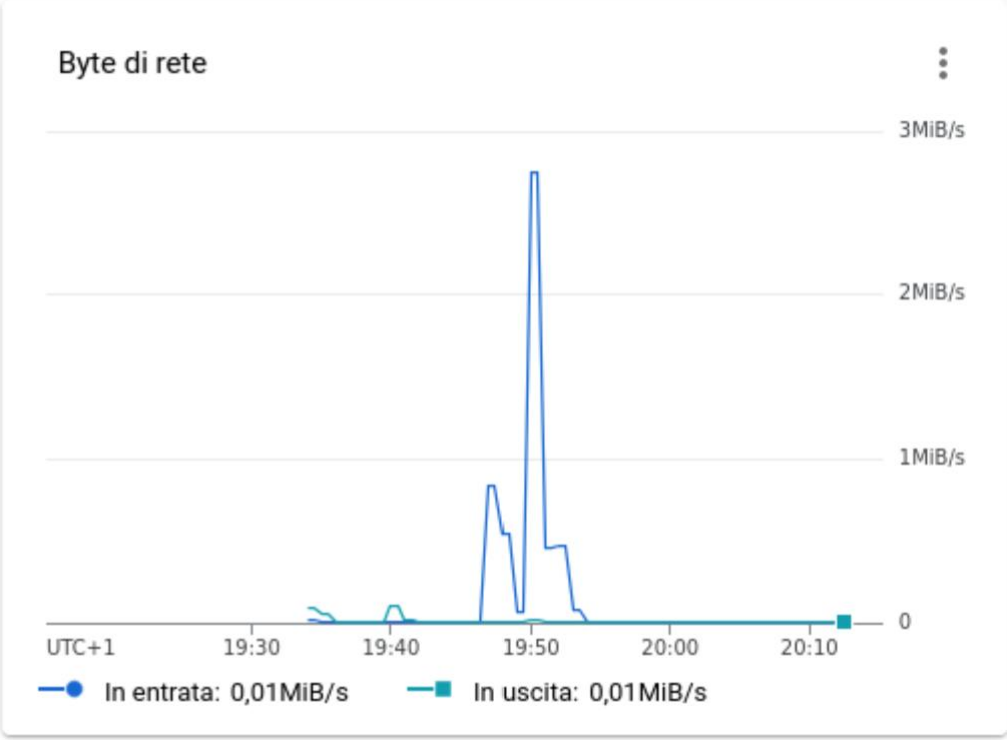


Fig5: byte di rete, Ray, configurazione 2.

5. Limiti e possibili estensioni

A causa delle tempistiche non è stato possibile implementare un cluster Ray analogo a quello Spark, ovvero composto da 1 master e 3 worker, poiché GCP non prevede un'implementazione abbastanza lineare nel caso di questo framework, ma sicuramente il confronto sarebbe notevolmente appropriato.

Inoltre si potrebbe implementare un cluster in locale (Standalone, Hadoop cluster) per capire fino a che punto tale implementazione superi in termini di prestazioni quella in rete.

Infine, si potrebbe estendere il progetto facendo utilizzo di dataset molto più grandi in termini di dimensioni, sui quali possono essere fatte operazioni molto più onerose in termini computazionali.