Calcolo redshift fotometrico su diversi cluster EMR di AWS

Carlo Cabras

Sommario

In questo lavoro cerco di calcolare lo spostamento verso il rosso di una sorgente astronomica basandomi su dati fotometrici, presenti nel dataset DR16 della SDSS composto da 5.107.045 sorgenti tra stelle, galassie e quasar.

Utilizzo metodi di regressione, in particolare *regressione lineare* per quanto riguarda un metodo ordinario, mentre *regressione su alberi decisionali* e *regressione su random forest* per quanto riguarda metodi basati su tecniche di *machine learning*, usando gli strumenti forniti da Apache Spark.

Per fare ciò, utilizzo un *cluster EMR* della piattaforma AWS di Amazon, creato tramite Terraform, testando combinazioni con 2, 4 e 8 nodi *core/slave* di tipo `m4.large` o `c4.large`, così da cercare eventuali differenze in termini di prestazioni tra le varie configurazioni.

Testo i metodi di regressione sia sul dataset intero che su diverse porzioni dello stesso, in modo da capire in quali classi di sorgenti funzionino meglio e peggio.

Si osservano miglioramenti nelle prestazioni nei cluster con istanze `c4.large` passando da 2 a 4 nodi, decisamente meno passando da 4 a 8. L'aumento dei nodi con istanze `m4.large` non dà un guadagno rilevante, se non per un caso isolato usando random forest con 4 istanze.

I modelli predicono male le categorie, sopratutto le stelle dove si hanno errori grandissimi. Si ottengono i risultati migliori — ma comunque da non ritenere buoni — in un dataset composto solamente da galassie e quasar. Alla luce dei risultati ottenuti e dei dati studiati, i metodi basati su machine learning utilizzati qua possono essere utili come base per una qualche ottimizzazione, ma non un metodo di predizione sicuro ed affidabile.

L'intero progetto si può trovare nella mia repository GitHub:
`https://github.com/carlocabras21/SpectralRegressionSpark`.

Indice

1 Contesto scientifico

1.1 Lo spostamento verso il rosso, o *redshift*

È da più di 90 anni che sappiamo che l'universo è in espansione: da quando Edwin Hubble correlò la distanza di nebulose extragalattiche con la loro velocità di recessione [2]. Oggi la legge che porta il suo nome afferma che esiste una relazione lineare tra lo spostamento verso il rosso della luce emessa dalle galassie e la loro distanza. Tanto maggiore è la distanza della galassia e tanto maggiore sarà il suo spostamento verso il rosso.

Lo spostamento verso il rosso è il fenomeno per cui la luce o un'altra radiazione elettromagnetica emessa da un oggetto in allontanamento ha una lunghezza d'onda maggiore rispetto a quella che aveva all'emissione. Ciò equivale a dire che nel caso della luce il colore va nella direzione dove è il rosso, l'estremo inferiore dello spettro del visibile. In generale, che la radiazione elettromagnetica sia visibile o meno, un redshift significa un aumento della lunghezza d'onda, equivalente a una diminuzione della frequenza. Quindi se ci aspettiamo che una sorgente emetta luce di una particolare lunghezza d'onda, quando essa si sta allontanando da noi quella lunghezza d'onda per *effetto Doppler* ci apparirà più lunga.

Il redshift di una sorgente astronomica è importante per poterne ricavare la distanza, usando la Legge di Hubble.

1.2 La fotometria

In astronomia, la fotometria è una tecnica che riguarda la misurazione dell'intensità della radiazione elettromagnetica di una sorgente celeste. Nella sua versione più basilare, la fotometria riguarda la rilevazione della luce passante per dei filtri ottici passabanda. Nel sistema fotometrico usato attualmente (che è più che altro uno standard *de facto*) si usano bande identificate da lettere, ad esempio il sistema "ugriz" riguarda le bande relative al vicino ultravioletto (u), il verde (g), il rosso (r) ed il vicino infrarosso (i, z). Si identifica come "colore" la differenza in magnitudine (luminosità) tra due filtri.

2 Scopo del progetto

Una galassia emette luce con uno spettro che è dato dalla summa di tutte le stelle che la compongono. La chiave nella classificazione dello redshift da dati fotometrici sta nel fatto che una galassia *redshiftata* avrà colori diversi rispetto a quando non lo è. Ma non è così semplice: se i colori sono diversi, come sappiamo se questa differenza è dovuta al redshift oppure è semplicemente una galassia che ha un diverso spettro?

Prendiamo in considerazione stelle, galassie e quasar (galassie così lontane da apparirci come puntiformi e con uno spettro molto redshiftato). Tenendo a mente la Legge di Hubble descritta poc'anzi, questi tre oggetti celesti avranno redshift completamente diversi tra loro. Le stelle, relativamente vicine, avranno

un redshift vicino allo zero, le galassie avranno un redshift più grande ed i quasar, oggetti che stanno praticamente ai confini dell'universo, avranno un redshift ancora maggiore.

Lo scopo di questo progetto è quello di applicare tecniche di Machine Learning al fine di calcolare il redshift delle suddette sorgenti celesti, a partire dai soli dati fotometrici. Si valuteranno inoltre le prestazioni di diversi tipi di cluster EMR di AWS.

3 Struttura del progetto

La cartella principale del progetto è `SpectralRegressionSpark`[1]. Essa contiene tutti i file Terraform e lo script Python con il programma vero e proprio, `SpectralRegressionSpark.py`.

All'interno della cartella `resources` troviamo lo script `extract_test.py` per estrarre una piccola porzione del dataset, ed il file `query.sql` contenente le query utilizzate per scaricare il dataset ed estrarre alcune informazioni. Una volta scaricato il dataset (come mostrato nella sezione 4.1), esso dovrà essere posto all'interno di questa cartella.

All'interno della cartella `results` troviamo i risultati dei miei test, suddivisi per categoria come spiegato nelle sezioni successive.

Io ho lavorato su Linux Mint, una distribuzione Linux basata su Ubuntu, pertanto spiegherò come far funzionare il progetto su Linux. È possibile che alcuni dei passi successivi (come la connessione SSH) siano diversi in altri sistemi operativi.

4 Il dataset

Il dataset utilizzato è la Data Release 16 [1] (DR16), quarta pubblicazione di dati della quarta fase della Sloan Digital Sky Survey (SDSS-IV). Di questi dati io uso solo i dati relativi alle osservazioni nelle bande ugriz (spiegate nella sezione 1.2), il tipo di sorgente (stella, galassia o quasar) ed il redshift.

Il dataset è composto da 5.107.045 record per un totale di 304 MB. Si hanno 1.041.130 stelle, 2.963.274 galassie e 1.102.641 quasar (QSO).

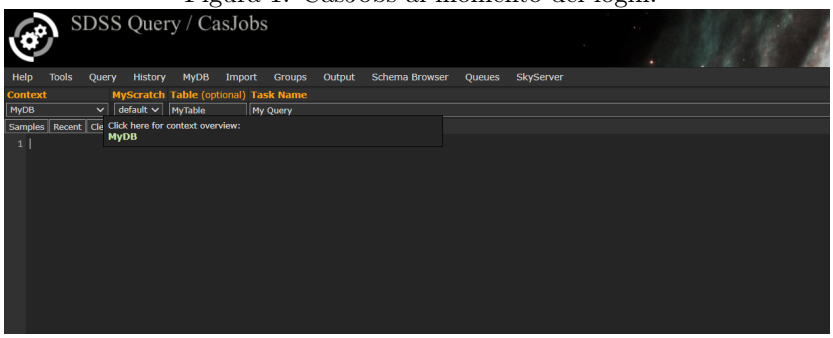
4.1 Download

Per poter scaricare il dataset, occorre andare su questo link
`http://skyserver.sdss.org/CasJobs/SubmitJob.aspx`
ed effettuare il login, eventualmente creando un account nel caso non se ne fosse in possesso. Se tutto è andato bene, ci ritroviamo davanti la schermata presente

[1] A fine progetto mi sono reso conto che un nome più appropriato sarebbe dovuto essere "PhotometricRegressionSpark" in quanto vi è una differenza tra le due parole.

in fig. 1.

Figura 1: CasJobs al momento del login.



Dal menu a tendina "Context", scegliere "DR16", come mostrato in fig. 2

Figura 2: Menù a tendina dove scegliere il dataset DR16.

Copiare ed incollare la seguente query[2]:

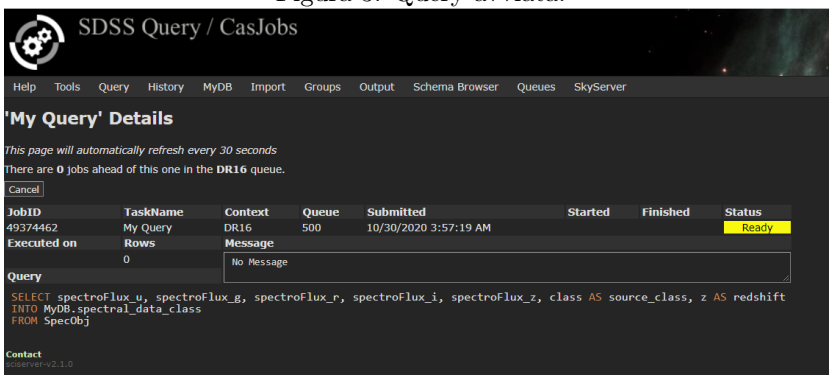
```
SELECT spectroFlux_u, spectroFlux_g, spectroFlux_r, spectroFlux_i,
    spectroFlux_z, class AS source_class, z AS redshift
INTO MyDB.spectral_data_class
FROM SpecObj
```

e lanciarla cliccando sul pulsante "Submit" presente sulla destra. Questa query ci salverà i dati che ci servono in un database personale che possiamo

[2]Tutte le query usate nel progetto sono presenti nel file `query.sql`.

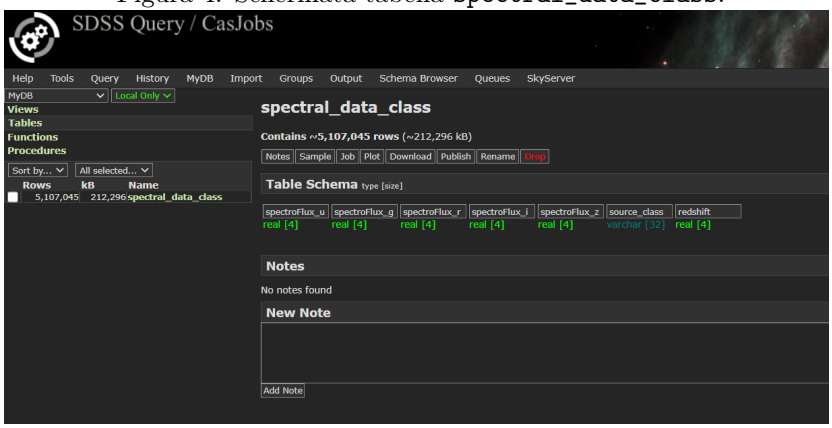
scaricare. Dopo aver premuto "Submit", la pagina i riporterà alla schermata che ci informa sull'avanzamento della query, come mostrato in fig. 3.

Figura 3: Query avviata.



Ricaricando la pagina possiamo controllare in che stato si trova la query. Passerà quasi subito da "Ready" a "Started". Quando passerà a "Finished" possiamo cliccare su "MyDB" dove troviamo la tabella `spectral_data_class`, clicchiamoci sopra e ci saranno mostrati schema e le operazioni che possiamo fare, come mostrato in fig. 4.

Figura 4: Schermata tabella `spectral_data_class`.



Clicchiamo su "Download", selezioniamo "Comma Separated Values" dal menu a tendina indicante il formato del file, e premiamo "Go"; fig. 5.

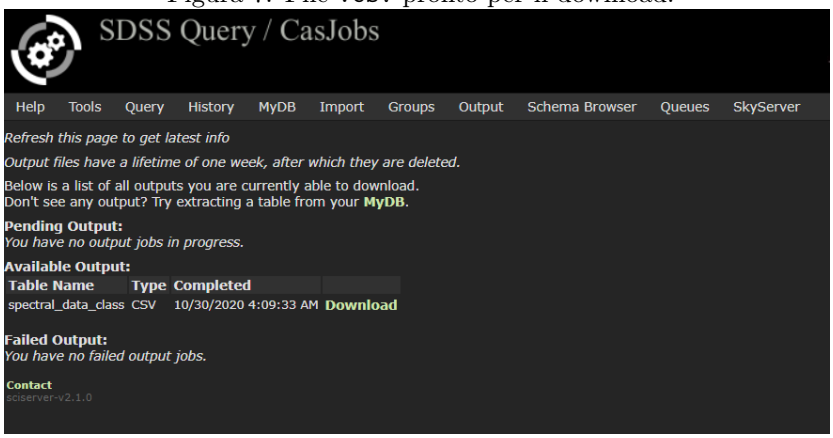
Figura 5: Download del dataset.

L'operazione precedente ci riporta alla pagina di output, dove attendiamo che il file `.csv` sia pronto, fig. 6.

Figura 6: Pagina di output durante la preparazione del file `.csv`.

Dopo qualche minuto il file è pronto e ricaricando la pagina ci comparirà il pulsante per il download, come mostrato in fig. 7.

Figura 7: File `.csv` pronto per il download.



Clicchiamo su "Download" e salviamo il file.

5 AWS e Terraform

Amazon Web Services, Inc. (AWS) è un'azienda statunitense di proprietà del gruppo Amazon, che fornisce servizi di cloud computing, tra i quali Amazon Elastic Compute Cloud (EC2), Amazon Elastic MapReduce (EMR) e Amazon Simple Storage Service (S3).

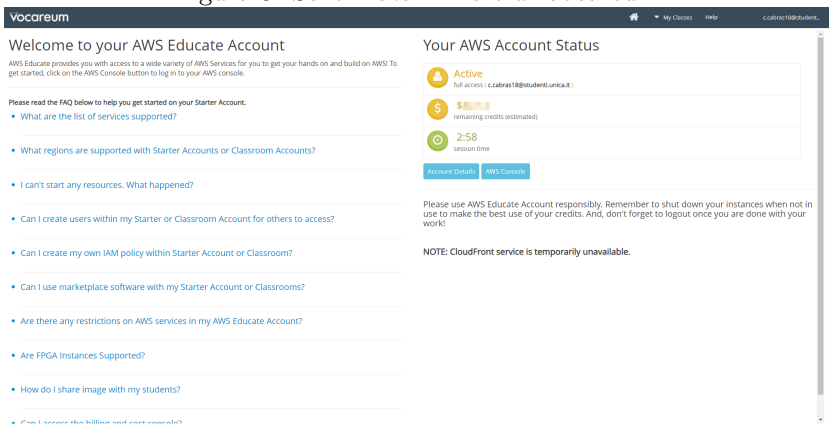
Terraform è un software del tipo *Infrastructure as code (IaC)* che permette di gestire infrastrutture di data center attraverso un linguaggio di configurazione dichiarativo conosciuto col nome di HashiCorp Configuration Language (HCL).

Nelle fasi iniziali del progetto ho usato semplici istanze EC2 — e non cluster di nodi — per conoscere AWS e Terraform. Non mostrerò come creare un'istanza ma passerò direttamente alla spiegazione degli script di Terraform per poter avviare un cluster EMR, con il caricamento dei dati in un bucket S3.

5.1 AWS Educate

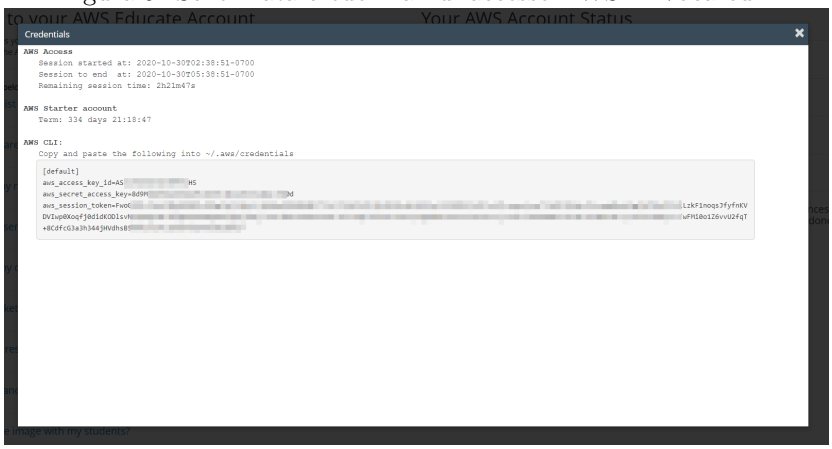
Lavorare su AWS costa, tuttavia possiamo sfruttare AWS Educate che per noi studenti ci consente di avere 100$ in crediti da spendere su AWS. Dopo essersi registrati (con l'email @studenti.unica.it) sul sito ed aver effettuato il login, cliccare in alto a destra su "AWS Account", poi nel bottone centrale "AWS Edcuate Starter Account". Ci troviamo davanti la pagina di Vocareum da cui possiamo controllare quanti crediti ci rimangono ed accedere ad AWS, come mostrato in fig. 8.

Figura 8: Schermata iniziale di Vocareum.



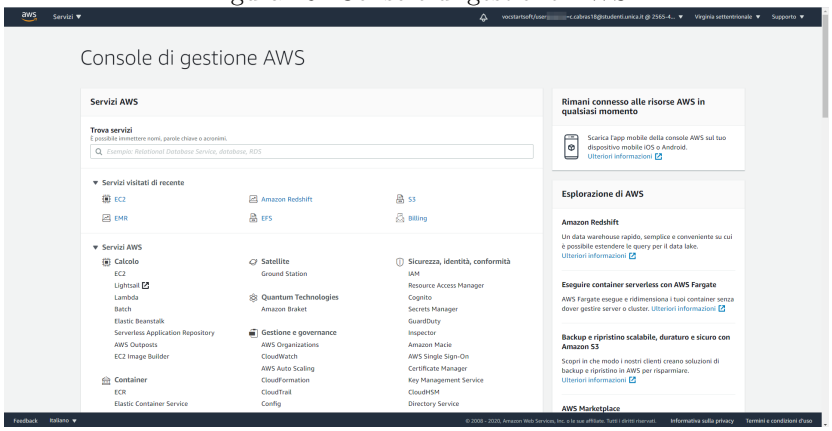
Cliccando su "Account Details", troviamo sotto la voce "AWS CLI" il pulsante "Show", che ci mostra un'importante pagina (fig. 9): quella con le credenziali di accesso. Queste dovranno essere inserite nello script Terraform in modo da garantire l'accesso ad AWS.

Figura 9: Schermata credenziali di accesso AWS in Vocareum.



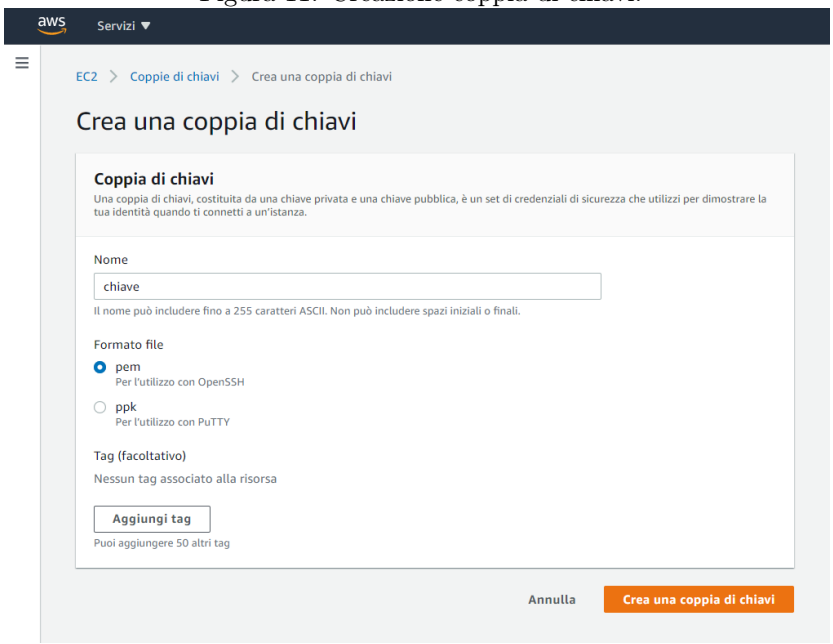
Tornando nella pagina iniziale di Vocareum, cliccando su "AWS Console" veniamo reindirizzati alla Console di gestione AWS, fig. 10.

Figura 10: Console di gestione AWS.



Ora dobbiamo creare una coppia di chiavi per l'autenticazione SSH. Da "Servizi" in alto a sinistra, andare su "EC2", cercandolo eventualmente nella barra di ricerca nel caso non sia tra i primi risultati. Nella sezione "Risorse" al centro della pagina, clicchiamo quindi su "Coppie di chiavi", poi su "Crea una coppia di chiavi". Inseriamo il nome, ad esempio "chiave", scegliamo come formato "pem" e clicchiamo su "Crea una coppia di chiavi" (fig. 11).

Figura 11: Creazione coppia di chiavi.



Si aprirà subito una schermata che ci permette di salvare il file `chiave.pem`: salviamolo subito, in quanto è la chiave privata e non possiamo più scaricarla. Cambiare i permessi con `chmod 400 chiave.pem` in modo che solo il nostro utente possa leggerla. Fare attenzione di avere i permessi sulla partizione su cui si sta operando.

Nel caso perdessimo il file, occorre creare una nuova coppia di chiavi. Questo file lo useremo nello script Terraform e durante la connessione SSH.

5.2 Gli script Terraform

Per costruire un'infrastruttura attraverso Terraform, si devono scrivere le istruzioni in file formato `.tf`. Le istruzioni non seguono un ordine, quindi non dobbiamo preoccuparci in che posizione dei file dichiariamo le nostre risorse. Una risorsa è il componente più importante di Terraform, in quanto descrive tutti gli oggetti facenti parte dell'infrastruttura. All'avvio del comando di creazione dell'infrastruttura, terraform leggerà tutti i file `.tf` presenti all'interno della cartella. Nel mio progetto mantengo le varie risorse in diversi file per mantenere un ordine logico, tuttavia non c'è bisogno di mantenerle separate e possiamo creare un file unico.

In questa relazione spiego brevemente il funzionamento delle risorse. Non riscriverò tutto il codice qua, in quanto per poterlo eseguire si può tranquillamente lavorare sui file presenti nella repository git. Come far partire gli script è spiegato nella sezione 5.2.6.

5.2.1 Variabili di accesso

Nel file chiamato `access_variables.tf` vanno inserite le credenziali mostrate nella pagina di Vocareum (vedi fig. 9), salvate come variabili. Queste credenziali cambiano quando scade la sessione o quando ricarichiamo la pagina, pertanto nel caso avessimo il bisogno di eseguire gli script Terraform, dobbiamo controllare che le credenziali siano aggiornate. Esse devono stare private, non dobbiamo condividerle con nessuno e, nel caso usassimo sistemi di versionamento come git, facciamo attenzione a cosa rilasciamo pubblicamente.

Il file si presenterà più o meno così:

```
variable "access-key" {
  type    = string
  default = "..."
}
variable "secret-key" = "..."
variable "token" "..."
variable "key_name" {
  type    = string
  default = "prima" // nome della chiave SSH
}
```

5.2.2 Il cluster

Il cluster è costruito nel file chiamato `cluster.tf`. All'inizio del file, dichiariamo il provider `aws` che serve per interagire con le risorse supportate da AWS. Indichiamo la regione dove si troverà il cluster e le credenziali di accesso, quelle che abbiamo salvato in un file esterno. Nel mio caso come regione ho scelto `us-east-1` perché era la default, ma possiamo scegliere una delle qualsiasi regioni che abbiamo a disposizione. L'importante è che il nostro account ed i tipi di istanze siano accessibili da quella regione.

Creiamo poi un ruolo IAM, che sarebbe una specie di amministratore, attraverso la risorsa `aws_iam_role`. Nelle risorse successive gli daremo i permessi necessari per poter operare nelle diverse istanze.

Creiamo quindi il cluster attraverso la risorsa `aws_emr_cluster`, dandogli un nome, la versione di emr che vogliamo utilizzare, le applicazioni da installare e l'indirizzo del bucket S3 dove verranno salvati i file di log. Per conoscere le versioni EMR disponibili, possiamo giochicchiare con la GUI di AWS, creando un nuovo cluster e vedendo le opzioni presenti in "Configurazioni del Software". Io ho scelto la versione 5.21.2 perché contiene una versione 2.x di Spark, la stessa installata nel mio computer. Consiglio di avere versioni uguali in modo da poter fare test veloci sulla propria macchina.

L'attributo `ec2_attributes` ci consente di definire la chiave RSA di accesso, i gruppi di sicurezza per istanza master ed istanze slave ed il profilo IAM. Dopodiché, passiamo alla costruzione dei nodi attraverso l'attributo `master_instance_group` dove indichiamo che tipo di istanza deve essere il nodo master, e `core_instance_group`, dove indichiamo sia il tipo di istanza che il numero dei nodi.

Infine, voglio che alla creazione del cluster mi vengano stampati l'ID del cluster e l'indirizzo pubblico a cui connettermi via SSH, quindi utilizzo il comando `output` per poterli vedere. Questi dati si possono sia vedere dalla GUI, sia richiamare tutte le volte che vogliamo attraverso il comando `terraform output`.

Dichiariamo inoltre alcuni dati relativi alle istanze EC2, come i gruppi di sicurezza, la chiave SSH di accesso e la sottorete, all'interno dell'attributo `ec2_attributes`.

5.2.3 Configurazione della rete

In `networks.tf` sono presenti le risorse per poter configurare una VPC (Virtual Private Cloud), che ci permette di creare una nostra sottorete privata con le risorse AWS di cui abbiamo bisogno. Dichiariamo la VPC tramite la risorsa `aws_vpc`, assegnando a `cidr_block` l'estensione della rete, e attivando il DNS.

Creiamo poi le sottoreti collegandole alla VPC sopradichiarata, e colleghiamo un gateway in modo da permettere il traffico in ingresso, il tutto fatto usando le risorse `aws_subnet` e `aws_internet_gateway`.

Creiamo infine le tabelle di routing e le colleghiamo alla rete e sottorete, tramite `aws_route_table` e `aws_route_table_association`.

5.2.4 Configurazioni gruppi di sicurezza

Dentro il file `security.tf`, sono presenti le configurazioni relative ad i gruppi di sicurezza per le istanze di tipo master e slave. L'istanza master deve essere in grado di comunicare con l'esterno, aprendo quindi le porte giuste, e con le istanze slave; queste ultime devono poter comunicare solo tra di loro.

5.2.5 Configurazione bucket S3

All'interno di un bucket S3 carichiamo il dataset ed alla fine dell'esecuzione del programma troveremo i risultati.

La configurazione si trova all'interno del file `s3_bucket.tf`. Ci sono sostanzialmente due tipi di risorse: `aws_s3_bucket`, dove al suo interno dichiariamo un nome per il bucket che dovrà essere univoco a livello globale, e `aws_s3_bucket_object` che ci permette di caricare un oggetto. Infine ci sono poi le risorse relative alla VPC che ci permettono di collegare il bucket al cluster e di potervi accedere.

5.2.6 Esecuzione

Aprendo una console nella cartella dove sono presenti i file Terraform, eseguire gli script lanciando il comando `terraform apply`. Dopo che ci mostra un riassunto di ciò che verrà creato, accetterà soltanto `yes` come risposta. Digitiamolo[3]. Dopo aver finito, mostrerà l'indirizzo IP a cui connettersi tramite SSH. Connettersi con

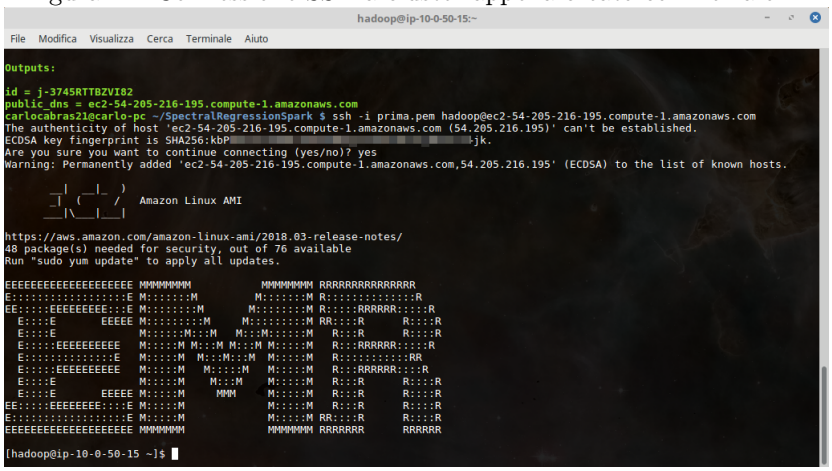
```
ssh -i <key-name>.pem hadoop@<public_dns>
```

dove `<key-name>` è il nome della chiave che abbiamo creato in precedenza e `public_dns` è il valore sputato fuori alla fine della creazione del cluster. `hadoop` è il nome utente.

Nella fig. 12 il risultato, dove la chiave utilizzata si chiamava `prima`.

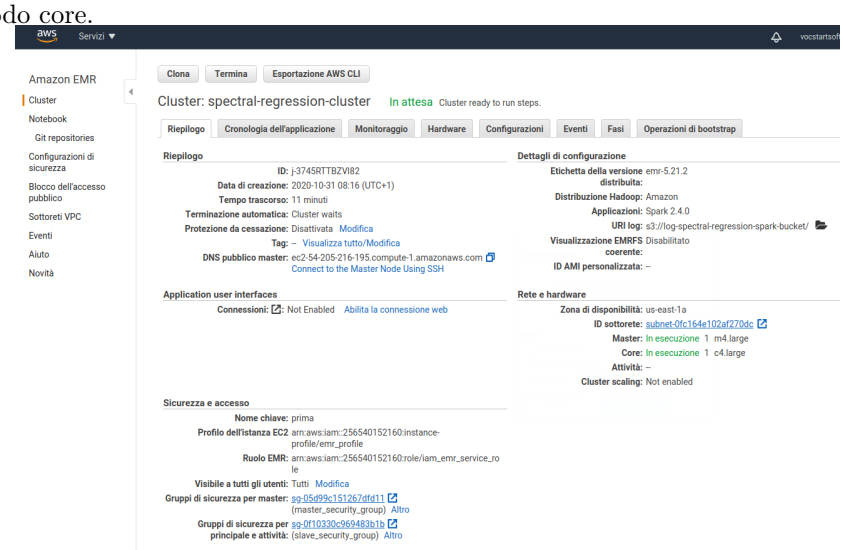
[3] Lanciando il comando `terraform apply --auto-approve` evitiamo che Terraform ci chieda di digitare `yes`, continuando in automatico.

Figura 12: Connessione SSH a cluster appena creato con Terraform.



Dalla GUI, il cluster si presenta come in fig. 13. Possiamo notare come l'ID e l'indirizzo DNS pubblico master siano gli stessi mostrati nell'output, quindi la GUI è un altro modo per ricavare quei dati. Troviamo gli stessi dati che avevamo inserito negli script Terraform, come i nomi del bucket di log o dei gruppi di sicurezza.

Figura 13: Il cluster EMR visto dalla GUI di AWS, in questo caso con 1 solo nodo core.



6 Tipi di test effettuati

Per il calcolo del redshift tramite regressione a partire da dati fotometrici, ho usato diversi metodi, tra cui regressione lineare, regressione basata su alberi decisionali e su random forest. L'errore utilizzato è il RMSE (root-mean-square error).

Questi metodi li ho testati su diverse configurazioni di cluster: una istanza `m4.large` come master, poi 2, 4 e 8 slave di tipo `m4.large` e `c4.large`. Le istanze di tipo `m4` sono generiche, mentre le `c4` sono ottimizzate per il calcolo. Volevo vedere se si notassero differenze prestazionali.

Il dataset ha tre categorie di sorgenti: stelle, galassie e quasar. Ho effettuato alcuni test usando solo due categorie (stelle-galassie, stelle-quasar, galassie-quasar) studiando quanto i metodi fossero in grado di separare le due categorie.

7 Lo script `SpectralRegressionSpark.py`

In `SpectralRegressionSpark.py` si trova tutto il codice PySpark utilizzato per eseguire il lavoro. È suddiviso in parti: setup dei flag, inizializzaione di Spark, caricamento dei dati, regressione e presentazione dei risultati. I dati sono scritti su S3 in un file chiamato `PART-00000` all'interno di una cartella chiamata `results_<slaves>_<regression-type>_<date-time>`, dove `<slaves>` indica che nodi sono stati utlizzati, `<regression-type>` che metodo di regressione, mentre `<date-time>` la data e l'orario in cui si sta creando la cartella.

Nella parte iniziale del setup si trovano i flag che l'utente deve impostare, ed è l'unica parte del codice che va modificata. In particolare:

- `test`: impostare a `True` nel caso si voglia eseguire il lavoro usando una piccola porzione del dataset. Di default si trova un dataset con 999 sorgenti; nel caso se ne volesse una porzione diversa, usare lo script `extract_test.py`;
- `write_results_in_S3`: impostare a `True` se si vuole che i dati siano scritti all'interno del bucket S3;
- `two_classes_dataset`: `True` se si vuole eseguire il lavoro su due tipologie di sorgenti anziché su tutte e tre. Dopo aver impostato `True`, scegliere che valore dovrà avere la stringa `filter_type` nelle righe successive;
- `regression_type`: che tipo di regressione eseguire. Scommentare la riga riguardante quella che vogliamo scegliere;
- `test_on_single_classes`: `True` se si vuole calcolare RMSE in un test set composto da una sola sorgente, per tutte e tre le sorgenti;
- `slaves`: scrivere qua che tipo di slaves si sta usando. Ad esempio, `4c4large` se si stanno usando 4 istanze di tipo `c4.large`.

Al momento, su git si trova una versione del cluster con 4 istanze `c4.large`, mentre i flag sono impostati come:

```
test = False,
write_results_in_S3 = True,
two_classes_dataset = False,
regression_type = "decision-tree",
test_on_single_classes = True,
slaves = "4xc4large".
```

Lo script prosegue con l'inizializzazione di Spark ed il caricamento dati su Dataframe. Per la regressione, occorre calcolare i "colori", ovvero le differenze tra le luminosità dei vari filtri. Inoltre i metodi di Spark richiedono che le feature siano unite in un unico vettore; pertanto le prime parti del codice servono ad impostare i dati per le funzioni di regressione.

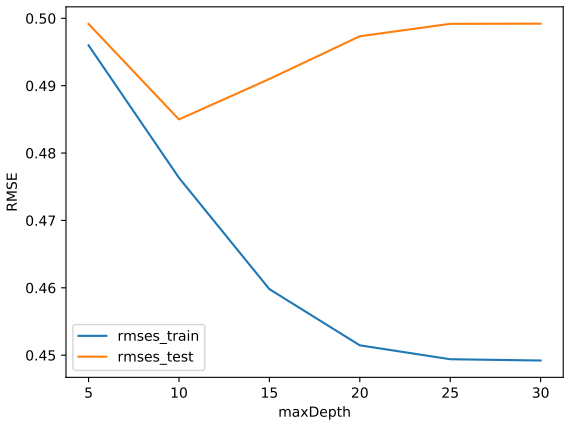
Si passa poi all'esecuzione dei metodi veri e propri ed alla stampa dei risultati nel bucket S3.

8 Tuning iniziale

I metodi di regressione usati hanno alcuni parametri da impostare. Ho effettuato questi test usando un dataset grande il 10% del totale.

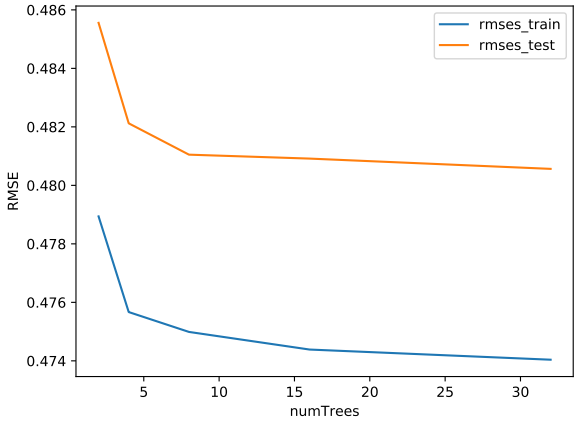
Per quanto riguarda gli alberi decisionali, possiamo giocare soltanto sulla sua profondità, cioè sul parametro `maxDepth`. I valori scelti sono stati pari a: $5, 10, 15, 20, 25, 30$. È risultato che il valore più basso per l'errore sui dati di test si ha per `maxDepth`=10, dove l'errore poi si alza leggermente e si ha overfitting sui dati di training, come mostrato in fig. 14.

Figura 14: Curva degli errori sui dati di training e di test al crescere della profondità degli alberi in Decision Tree Regression.



Su Random Forest, ho settato il numero degli alberi, cioè `numTrees`. In questo caso i valori scelti sono stati $2, 4, 8, 16, 32$. Osservando il grafico presente in fig. 15, ho scelto `numTrees`=8, in quanto gli errori al crescere di `numTrees` diminuivano di poco ma sarebbe aumentato il carico computazionale.

Figura 15: Curva degli errori sui dati di training e di test al crescere del numero di alberi in Random Forest Regression.



In entrambi i casi però non sono potuto andare oltre questi numeri, in quanto Spark non me lo permetteva. Ritengo che sia dovuto probabilmente al fatto che gli sviluppatori fossero consci del fatto che al crescere di questi numeri si ottiene overfitting. Infatti anche dai grafici si nota come gli errori varino di pochissimo. Ad esempio all'aumentare della profondità degli alberi si dovrebbe ottenere un notevole overfitting sui dati di training e underfitting sui dati di test. Qua non è stato possibile ottenere questo risultato perché i parametri non potevano crescere più di tanto.

9 Esecuzione su cluster EMR

Dopo aver creato il cluster ed aver caricato i dati tramite Terraform, dobbiamo eseguire lo script. Ci sono diversi modi per farlo, illustro quello che ho usato di più.

Trovandosi i dati su S3, dobbiamo solo inviare lo script al cluster. Possiamo caricare anche questo su S3, ma era una soluzione sconveniente perché capitava spessissimo di dover applicare piccole modifiche e caricare ogni volta il file su S3 era troppo dispendioso. Quindi ho deciso di inviare il file via SSH tramite il comando `scp`. Questo comando ha una struttura del tipo:
`scp -i chiave.pem <file> hadoop@<public_dns>:<path destinazione>`
Nel mio caso, inviavo il file nella directory `home` del cluster, in modo da avercelo

subito comodamente. Quindi:
`scp -i prima.pem SpectralRegressionSpark.py hadoop@<public_dns>:~/`

Una volta inviato il file, ci possiamo connettere al cluster tramite SSH e far partire il programma tramite
`spark-submit --deploy-mode cluster SpectralRegressionSpark.py`
L'applicazione ci metterà diversi minuti per completare. A video vedremmo aggiornarsi ogni secondo il suo stato, che è "RUNNING". Una volta che finisce vedremmo un messaggio con "SUCCEEDED". Nel bucket S3 troveremo i risultati in una cartella formattata come spiegato nella sezione 7.

Una volta terminate l'applicazione, possiamo leggere i log di output attraverso il comando:
`yarn logs -applicationId <your applicationId> > log.txt`
dove l'ID dell'applicazione è qualcosa tipo `application_1603873119012_0005` che vediamo facilmente durante la sua esecuzione.

IMPORTANTE

Una volta svolte tutte le operazioni, ricordarsi di distruggere il cluster, in modo da evitare di consumare crediti inutilmente. Si può fare con un comando Terraform, ma prima occorre svuotare manualmente i bucket S3 dalla GUI, in quanto sono stati modificati e Terraform non li può più distruggere. Una volta fatto, lanciamo:

```
terraform destroy --auto-approve
```

dove anche in questo caso l'opzione `--auto-approve` avvia l'operazione senza attendere conferma.

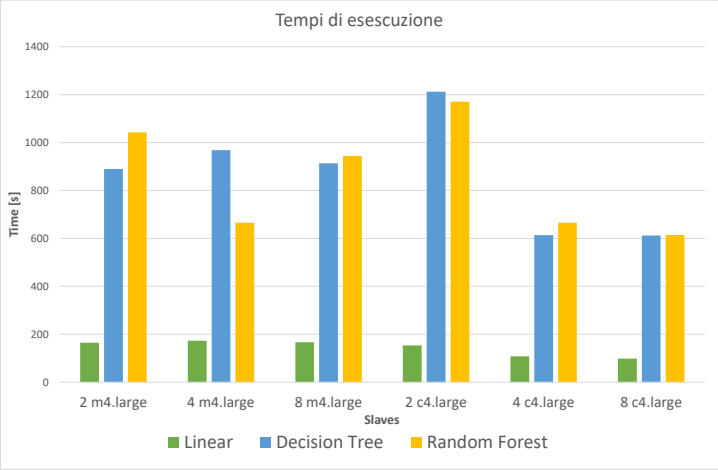
10 Riassunto delle operazioni da effettuare

- Scaricare il dataset e salvarlo nella cartella `SpectralRegressionSpark/resources`.
- Prendere le credenziali da Vocareum e scriverle nel file `access_variables.tf`.
- In `cluster.tf` impostare quali istanze si vogliono come master e come core. Nel caso si volesse ripetere uno dei miei test, la master dev'essere una `m4.large` mentre i core devono essere 2, 4 o 8 `m4.large` o `c4.large`.
- Lanciare `terraform apply --auto-approve` per far partire la creazione del cluster e di tutte le risorse.
- Verificare l'avvenuta creazione connettendosi via SSH tramite `ssh -i chiave.pem hadoop@public_dns`
- Tornati in locale, settare i flag dello script ed inviare lo script al cluster tramite:
  `scp -i chiave.pem SpectralRegressionSpark.py hadoop@<public_dns>:~/`
  e, dopo essersi riconnessi al cluster, farlo partire con:
  `spark-submit --deploy-mode cluster SpectralRegressionSpark.py`
- Nel bucket S3 si troveranno i risultati nella cartella `results_<slaves>_<regression-type>_<date-time>`.
- Infine, svuotare i bucket manualmente dalla GUI e distruggere il cluster e tutte le risorse allocate tramite
  `terraform destroy --auto-approve`

11 Discussione risultati e conclusioni

Per valutare le prestazioni dei diversi cluster, confronto qui i tempi di esecuzione dello script. Le istanze m4 (di tipo generico) dovrebbero essere più lente di quelle c4 (ottimizzate per il calcolo). Inoltre, dato il parallelismo di Spark, ci si aspetta che all'aumentare dei nodi si ottengano tempi inferiori.

Figura 16: Confronto dei tempi di esecuzione.



Possiamo notare come la regressione lineare sia molto più veloce rispetto ai metodi di machine learning, e questo c'era da aspettarselo visto che si tratta di fitting dei dati su una retta. La velocità però si ripaga con l'errore, che è il più alto di tutti quanti i metodi (vedi più avanti).

Osservando i tempi dei cluster con core m4, non si notano sostanziali miglioramenti all'aumentare dei nodi. Si nota un valore molto basso riguardante Random Forest nel caso di 4 m4.large che può suonare come un caso particolare, ma ho ripetuto il test ed il tempo era lo stesso.

Mentre per quanto riguarda le istanze c4, passando da 2 a 4 core i tempi scendono decisamente, mentre passando ad 8 non si ha un miglioramento rilevante. Come rapporto costi/benefici, direi che la configurazione migliore sia quella con 4 c4.large.

Per quanto riguarda gli errori ottenuti nei diversi metodi di regressione, ho usato come stima RMSE (root-mean-squared error), ottenendo mediamente questi valori:

	Linear Regression	**Decision Tree**	**Random Forest**
RMSE	0.841882286	0.683437032	0.676951173

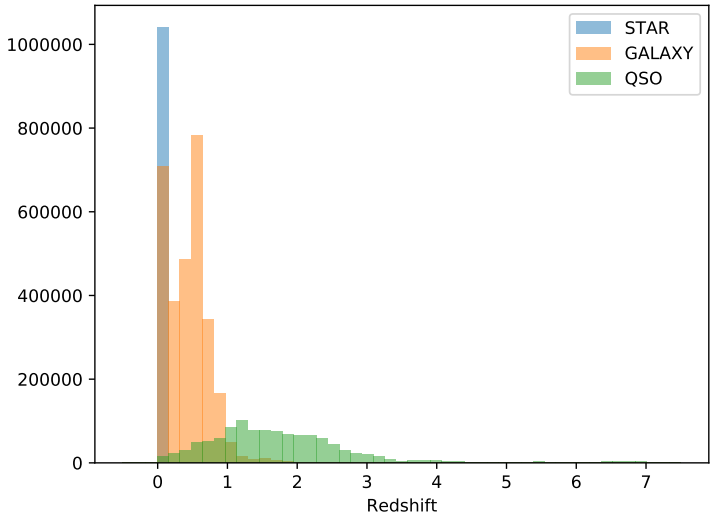
Il valore RMSE va rapportato al valore dei redshift dei record. Cerchiamo prima di capire come si presentano i dati, osservando per ogni categoria il valore medio del redshift (indicato come **z**) con la sua deviazione standard, ed i valori minimi e massimi:

Class	**avg(z)**	**stdev(z)**	**min(z)**	**max(z)**
STAR	-0.00010	0.000817	-0.00504	0.00506
GALAXY	0.43176	0.28636	-0.011447	2.00262
QSO	1.77509	1.12742	-0.00076	7.05564

Da questa tabella si può notare numericamente il discorso affrontato nella sezione 2, dove dicevo come il redshift aumenta all'aumentare della distanza degli oggetti.

In un istogramma possiamo inoltre avere un'idea della distribuzione:

Figura 17: Istogramma dei dati.



L'errore medio è alto per tutte le categorie dei dati. Vediamo quindi gli errori sulle singole categorie di dati:

	Linear	**Decision Tree**	**Random Forest**
STAR	0.633621357	0.578758244	0.577501186
GALAXY	0.350755162	0.26376567	0.25611034
QSO	1.599488819	1.278060947	1.280820703

Per quanto riguarda le stelle, gli errori sono esageratamete alti. I valori del redshift delle stelle nel dataset vanno da -0.00504 a 0.00506, mentre gli RMSE ottenuti vanno da 0.58 a 0.63: questo valore implica che ad esempio il modello confonda tra stelle e galassie.

Le galassie hanno un redshift medio di 0.43176, con deviazione standard di 0.28636, e spaziano tra -0.011447 e 2.00262. L'errore ottenuto è circa 0.35 per quanto riguarda la regressione lineare, 0.26 per i metodi su machine learning. Prendendo in considerazione quest'ultimo visto che è il migliore, non sono sicuramente errori gravi come i precedenti, ma neanche trascurabili. Posso considerarli come punto di partenza per eventuali ottimizzazioni.

I quasar hanno mediamente redshift di 1.77509 con deviazione standard di 1.12742, spaziando tra -0.00076 e 7.05564; sono quindi quelli che hanno il più grande range di valori. Gli errori ottenuti sono nell'ordine di 1.6 per la regressione lineare, mentre 1.28 per i metodi di machine learning. Come prima, ritengo questi risultati un punto di partenza più che una conclusione.

Come ultima analisi, ho studiato le capacità che i modelli hanno di separare due classi, prendendone in esame due alla volta. Partiamo con stelle e galassie:

	Linear	**Decision Tree**	**Random Forest**
STAR	0.319595376	0.194203166	0.200027798
GALAXY	0.308374358	0.16332472	0.165040018
media	0.311329311	0.171899662	0.174793481

Errori un minimo accettabili per le galassie, ma non per le stelle.
Proseguiamo con stelle e quasar:

	Linear	**Decision Tree**	**Random Forest**
STAR	0.913455974	0.648196309	0.648938054
QSO	1.414097486	1.205358163	1.196596931
media	1.197371069	0.97666261	0.970807776

Stesso discorso precedente, il modello continua a sbagliare (e di molto) nel predirre le stelle. Concludiamo con galassie e quasar:

	Linear	**Decision Tree**	**Random Forest**
GALAXY	0.463562663	0.330436306	0.326782424
QSO	1.497742497	1.16267592	1.146855585
media	0.876041385	0.6669887	0.658664449

Questo è il caso in cui i modelli si comportano nel modo migliore, anche se non ancora in modo affidabile.

In conclusione, alla luce dei dati ottenuti, il cluster che ha avuto le prestazioni migliori è stato quello con 4 core di tipo `c4.large`, nonostante con 8 si sia comportato leggermente meglio. Salvo per un caso isolato di Random Forest su 4 istanze `m4.large`, con questa tipologia l'aumentare dei core non migliora le prestazioni.

I modelli di predizione non funzionano, o comunque non sono affidabili presi così da soli. La regressione lineare è veloce ma è quella con gli errori più grandi, mentre i modelli basati su machine learning non danno dei risultati che giustifichino la complessità computazionale.

Riferimenti bibliografici

[1] Ahumada et al. The sixteenth data release of the sloan digital sky surveys. *Astrophysical Journal Supplement Series*, 249(1), 2020.

[2] Edwin Hubble. A relation between distance and radial velocity among extra-galactic nebulae. *Proceedings of the National Academy of Sciences*, 15(3):168–173, 1929.