

# UNIVERSITA' DEGLI STUDI DI NAPOLI FEDERICO II

DIPARTIMENTO DI INGEGNERIA ELETTRICA E DELLE  
TECNOLOGIE DELL'INFORMAZIONE

LAUREA MAGISTRALE DI INGEGNERIA INFORMATICA

CORSO DI BIG DATA ANALYTICS AND BUSINESS  
INTELLIGENCE

## EXPERT FINDING

IDENTIFICAZIONE UTENTI CON CONOSCENZA SU UN DATO TEMA



*Autori:*

Davide LAURETANO

M63000792

Michele POMMELLA

M63000790

Davide TRIMALDI

M63000799

*Professore:*

Antonio PICARIELLO

ANNO ACCADEMICO 2018-2019

# Indice

1	Problema	5
2	Metodologia ed architettura	7
3	Risultati sperimentali	20

# Elenco delle figure

2.1	MongoDB Spark Connector . . . . .	8
2.2	Architettura . . . . .	9
3.1	MongoDB . . . . .	21
3.2	Esperti di golf . . . . .	22
3.3	Esperti di animali . . . . .	23
3.4	About Me degli esperti di animali . . . . .	23
3.5	Esperti di informatica . . . . .	25

# Listings

2.1	Yelp Dataset Loading . . . . .	10
2.2	StackExchange Dataset Loading . . . . .	10
2.3	Spark Packages . . . . .	12
2.4	Expert Finding . . . . .	14
2.5	Driver Memory . . . . .	18

# List of Algorithms

1	Expert Finding Pseudocode . . . . .	19
---	-------------------------------------	----

# Capitolo 1

## Problema

La ricerca di esperti è una sfida per molte ragioni, tra cui:

- Il volume di pubblicazioni di un esperto non è indice di esperienza;
- Il primo esperto che hai trovato potrebbe non essere il migliore;
- Alcuni argomenti generano più opinioni che fatti e quindi trovare il vero esperto può essere difficile;
- Di solito c'è una mancanza di accesso alle informazioni sulle prestazioni passate dell'esperto;
- Le competenze non sono distribuite in modo uniforme e punti di forza delle associazioni tra esperti variano in modo significativo
- Non ci sono standard che specificano i criteri e/o le qualifiche necessarie per particolari livelli di esperienza.
- La vera esperienza è rara e costosa. Spesso l'accesso è controllato, in modo informale o formalmente, dall'esperto stesso o dal loro management.
- La competenza di un esperto cambia continuamente e richiede consapevolezza di questa dinamica.
- Le soluzioni a problemi complessi spesso richiedono o comunità di esperti o diverse gamme di competenze che devono essere riunite per risolvere il complesso i problemi.
- Gli ingegneri di uno studio classico hanno trascorso il 16% del loro tempo a comunicare esperti - ma la comunicazione è stata ostacolata da differenze geografiche, di fuso orario e barriere culturali.

In sintesi, la ricerca di esperti è un compito complesso e difficile.

## Capitolo 2

# Metodologia ed architettura

La metodologia adoperata fa uso di un database *NoSQL*, per il mantenimento dei dati, e della tecnologia *Apache Spark* su macchina virtuale *Azure*.

La macchina virtuale mette a disposizione:

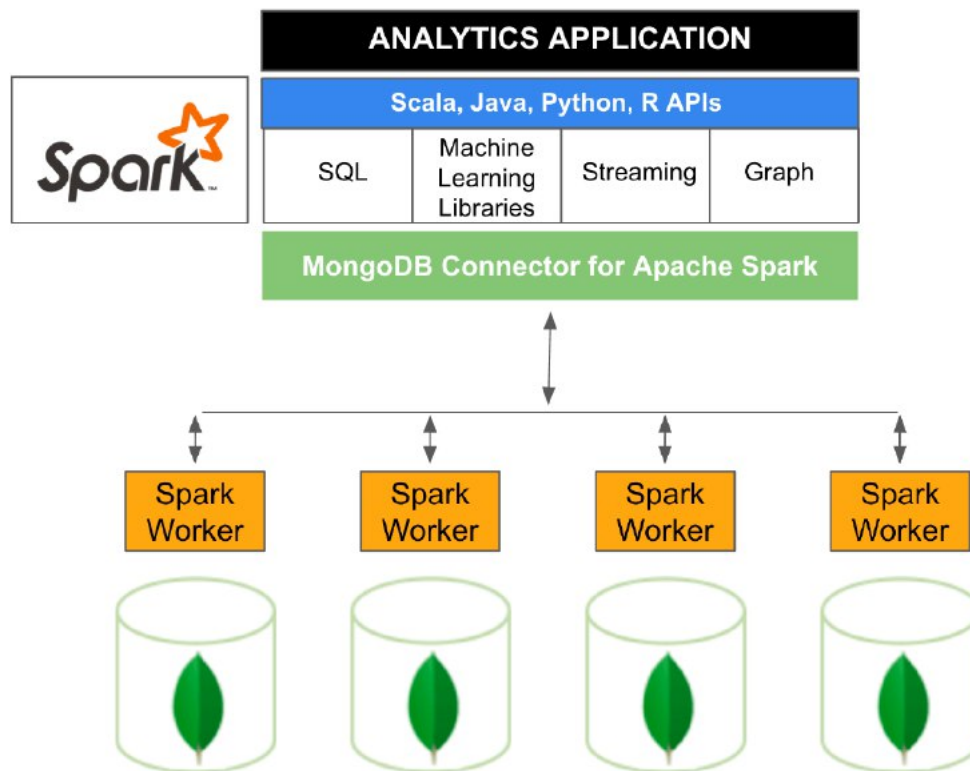
- 8 core
- Memoria centrale di 58 GB
- Memoria di massa di 30 GB
- Disco temporaneo secondario di 400 GB

Accessibile mediante protocollo SSH, offre l'utilizzo dell'engine Apache Spark con stack relativo già installato e configurato. In particolare si è fatto uso del modello di programmazione Spark per il linguaggio Python: *Pyspark*.

Il database NoSQL utilizzato è *MongoDB* per la tipologia di dati da trattare e le operazioni da effettuare su di essi. MongoDB è un database documentale che consente di trattare aggregati strutturati, risultando in accessi più flessibili. Permette, quindi, di sottomettere query basate sui campi dell'aggregato e recuperare solo parte di esso. MongoDB memorizza e restituisce documenti in formato *BSON*, rappresentazione binaria del JSON. I documenti sono strutture dati ad albero gerarchiche e possono essere anche strutturalmente non identici. Sono infatti distinti per collezione, insieme di documenti simili. Le caratteristiche di MongoDB hanno consentito una gestione semplice ed efficace del dataset composto principalmente da file *JSON* e *XML*. Il flessibile modello documentale dei dati con schema dinamico e ridimensionamento automatico su hardware comune rende MongoDB ideale per applicazioni con grandi volumi di dati multi-strutturati e dall'elevato tasso di cambiamento. Inoltre MongoDB offre la completa interoperabilità



con il sistema Spark, mediante il *MongoDB Spark Connector*, che ne consente la gestione tramite codice Pyspark.



**Figura 2.1:** MongoDB Spark Connector

Apache Spark è un framework di data processing che consente di effettuare query veloci grazie alla memorizzazione *in-memory* dei dati. Supporta diversi linguaggi di programmazione: Scala, Java, R, Python. Le sue principali caratteristiche sono:

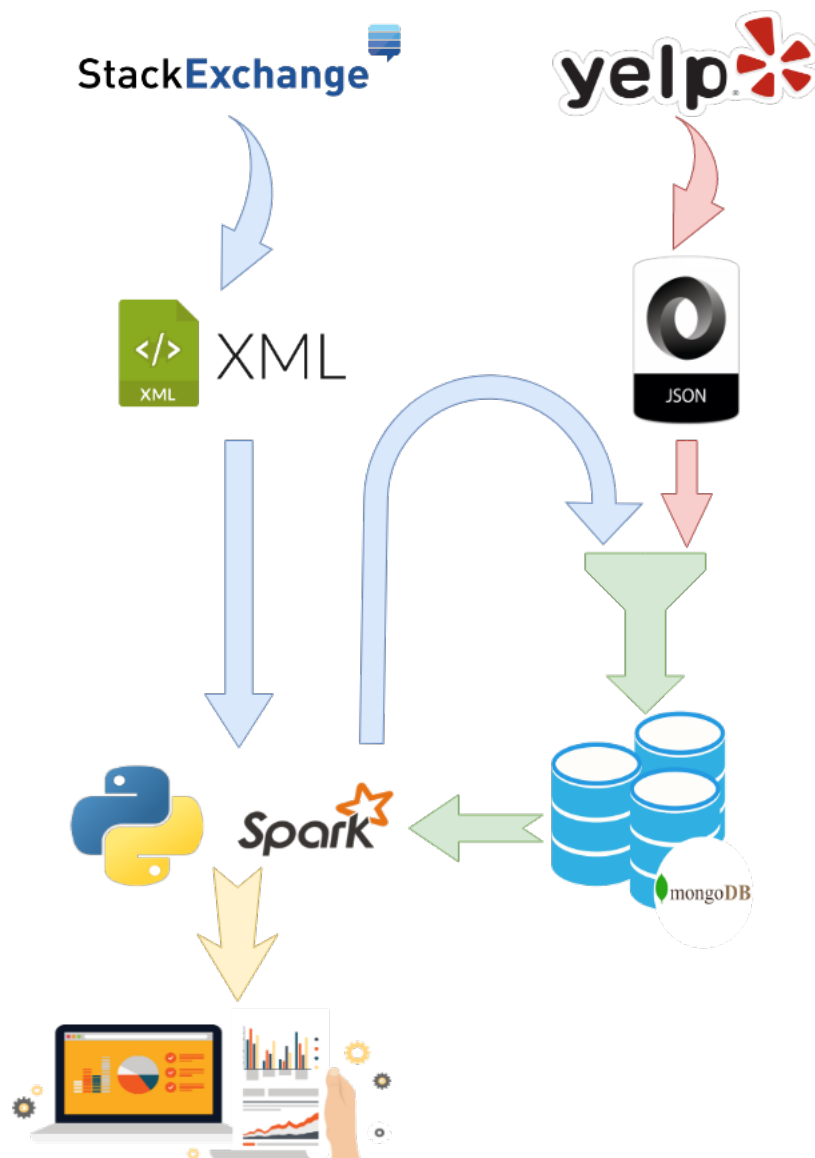
**Velocità** sfruttando le ottimizzazioni in-memory;

**Framework unificato** offrendo packages di librerie di alto livello (supporto a query SQL, Machine Learning, stream e graph processing);

**Semplicità** includendo API facili da usare per operare su grandi dataset, come operatori per trasformare e manipolare dati semistrutturati.

Spark, combinato con Python, è stato sfruttato sia per il caricamento del dataset di file XML nel database, non nativamente convertibile in BSON da MongoDB, sia per l'elaborazione dei dati.

L'unione di queste due tecnologie consente allo sviluppatore di realizzare l'applicazione più velocemente, utilizzando un solo database. Spark può eseguire direttamente sui dati operativi posti in MongoDB, senza tempi e costi di un processo di ETL. MongoDB può efficientemente presentare di ritorno i risultati analitici ai processi operativi.



**Figura 2.2:** Architettura

I file JSON di Yelp sono stati caricati direttamente in MongoDB grazie al JSON parser offerto. Per il caricamento, ad esempio, delle recensioni, si

è potuto adoperare il comando in **2.1**. Esso indica in quale database ed in quale collezione caricare i dati.

```
mongoimport --db BDABI --collection yelpreview --file review.json
```

**Listing 2.1:** Yelp Dataset Loading

Per i file XML, invece, MongoDB non offre delle estensioni per la conversione in BSON. Per il caricamento dei file di StackExchange, quindi, è stato adoperato Pyspark. Il codice prevede la creazione di una sessione Spark e la definizione degli schemi XML. Segue la creazione dei *DataFrame* a partire dai file XML. La definizione dello schema non è strettamente necessaria per il caricamento: Spark potrebbe effettuare l'analisi dell'intero dataset e ricavarne lo schema di conseguenza. Essa, quindi, mira ad efficientare l'elaborazione, dando a Spark indicazioni sullo schema da ricerca ed i tipi degli attributi relativi, evitando l'analisi completa del dataset. Ciò si rivela cruciale quando si gestiscono grandi volumi di dati. Nel caso di StackExchange è richiesto che Spark elabori e carichi circa 60GB di file XML appartenenti ai diversi topic. Dopo la definizione dei *DataFrame*, segue la scrittura in MongoDB degli stessi. La collezione in cui scrivere i dati è determinata dal parametro di ingresso alla funzione, rappresentante il topic. In tal modo i dati di StackExchange sono già differenziati per topic e sarà semplice un recupero in tal senso nella fase finale di analisi.

```
#coding=utf-8
from pyspark.sql import SparkSession
from pyspark.sql.types import *
import sys

#Funzione con parametro di ingresso: sys.argv[1] = topic

database = "BDABI"
collectionpost = "stackexchange"+sys.argv[1]+"post"
collectionuser = "stackexchange"+sys.argv[1]+"user"

#Configurazione SparkSession e connessione con MongoDB
spark = SparkSession.builder.appName("stackExchange")
    .config("spark.mongodb.input.uri",
            "mongodb://127.0.0.1/"+database+"."+collectionpost)
    .config("spark.mongodb.output.uri",
            "mongodb://127.0.0.1/"+database+"."+collectionpost)
    .getOrCreate()
```

*#Schemi dei file XML*

```
postSchema = StructType([StructField("_Id", StringType(), True),
    StructField("_PostTypeId", StringType(), True),
    StructField("_ParentId", StringType(), True),
    StructField("_AcceptedAnswerId", StringType(), True),
    StructField("_CreationDate", StringType(), True),
    StructField("_Score", IntegerType(), True),
    StructField("_ViewCount", IntegerType(), True),
    StructField("_Body", StringType(), True),
    StructField("_OwnerUserId", StringType(), True),
    StructField("_LastEditorUserId", StringType(), True),
    StructField("_LastEditorDisplayName", StringType(), True),
    StructField("_LastEditDate", StringType(), True),
    StructField("_LastActivityDate", StringType(), True),
    StructField("_CommunityOwedDate", StringType(), True),
    StructField("_ClosedDate", StringType(), True),
    StructField("_Title", StringType(), True),
    StructField("_Tags", StringType(), True),
    StructField("_AnswerCount", IntegerType(), True),
    StructField("_CommentCount", IntegerType(), True),
    StructField("_FavoriteCount", IntegerType(), True),
    StructField("_OwnerDisplayName", StringType(), True)])
```

```
userSchema = StructType([StructField("_Id", StringType(), True),
    StructField("_Reputation", IntegerType(), True),
    StructField("_CreationDate", StringType(), True),
    StructField("_DisplayName", StringType(), True),
    StructField("_AccountId", StringType(), True),
    StructField("_LastAccessDate", StringType(), True),
    StructField("_WebSiteUrl", StringType(), True),
    StructField("_Location", StringType(), True),
    StructField("_ProfileImageUrl", StringType(), True),
    StructField("_AboutMe", StringType(), True),
    StructField("_Views", IntegerType(), True),
    StructField("_UpVotes", IntegerType(), True),
    StructField("_DownVotes", IntegerType(), True),
    StructField("_Age", IntegerType(), True),
    StructField("_EmailHash", StringType(), True)])
```

*#Creazione DataFrame*

```
post = spark.read.format('xml').options(rowTag='row')
```

```

        .load( '/mnt/data/' + sys.argv[1] + '/Posts.xml', schema=postSchema )
        .withColumnRenamed( "_Id", "_postId" )
user = spark.read.format( 'xml' ).options( rowTag='row' )
        .load( '/mnt/data/' + sys.argv[1] + '/Users.xml', schema=userSchema )
        .withColumnRenamed( "_Id", "_userId" )

#Scrittura DataFrame in MongoDB
post.write.format( "com.mongodb.spark.sql.DefaultSource" )
        .mode( "append" )
        .save()
user.write.format( "com.mongodb.spark.sql.DefaultSource" )
        .mode( "append" )
        .option( "database", database )
        .option( "collection", collectionuser )
        .save()

```

**Listing 2.2:** StackExchange Dataset Loading

La funzione di caricamento del dataset di StackExchange adoperata per l'esecuzione i package messi a disposizione da Spark. In questo caso si rivela la potenza di Spark, che aumenta la produttività dello sviluppatore grazie alla sua interoperabilità con librerie di alto livello. In particolare sono stati utilizzati i package per la connessione mongo-spark e per l'analisi XML:

```

spark-submit --packages com.databricks:spark-xml_2.12:0.5.0,
org.mongodb.spark:mongo-spark-connector_2.12:2.4.0
/home/vmadmin/src/StackExchange.py topic

```

**Listing 2.3:** Spark Packages

MongoDB sarà ora pronto per fornire i dati per le elaborazioni successive. L'algoritmo, realizzato in Pyspark, si pone l'obiettivo di ricercare gli utenti esperti in un determinato argomento. Ricevuto in input il topic di riferimento, l'algoritmo effettua una ricerca nei diversi dataset e recupera i *top10* utenti che valuta come esperti. La valutazione della competenza degli utenti è effettuata mediante una metrica che si avvale di diversi parametri offerti dai dataset.

L'algoritmo, in prima battuta, trasforma il topic in input nella forma coerente con i diversi dataset. Essi, infatti, memorizzano i topic con formati diversi e questa eterogeneità è gestita a monte in tal modo: Yelp richiede una stringa in minuscolo con la prima lettera in maiuscolo ed uno spazio tra le diverse parole, StackExchange richiede una stringa tutta in minuscolo senza spazi tra le parole. Dopo la fase di trasformazione, segue la ricerca.

In Yelp si recuperano tutte le imprese appartenenti alla categoria ricercata e si sfruttano i loro id per trovare le recensioni degli utenti relativi mediante l'operazione di *join* offerta da Spark. Ad ogni recensione è associata un'informazione di utilità valutata dai lettori. Si filtrano, quindi, le recensioni con utilità maggiore della media e si ordinano in ordine decrescente per utilità. Si selezionano le 100 recensioni più utili. Infatti l'algoritmo restituirà i 10 utenti più esperti e sarebbe dispendioso continuare l'elaborazione considerando un numero di recensioni superiore di più di un ordine di grandezza rispetto al numero di utenti cercati. Tra le recensioni più utili si devono determinare gli utenti più esperti. Si procede, quindi, ad un ulteriore join con gli utenti. Si raggruppano le recensioni per utente e si somma l'utilità di quelle afferenti allo stesso utente, creando l'attributo di *topicuseful*. Esso rappresenta la somma cumulativa delle recensioni scritte da uno stesso utente relative al topic di riferimento. Le recensioni utili da sole non bastano ad identificare un utente esperto. Si è presa in considerazione il concetto di credibilità dell'utente, di autorità. Per fare ciò si sono prese in considerazione differenti informazioni circa l'utente: il numero di anni in cui è stato un utente elite (votato dagli altri utenti come competente), il numero di complimenti per le sue recensioni, la somma cumulativa dell'utilità delle sue recensioni, il numero di complimenti sul suo profilo, il numero di fan, il numero di complimenti per la lista di imprese recensite. Si è definita, dunque, una metrica per ricavare una valutazione della competenza dell'utente a partire da queste informazioni. Per renderle omogenee, esse sono state normalizzate nel range  $[0,1]$ :

$$x_{normalizzato} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Si è poi definita la metrica di valutazione della competenza come somma pesata di questi attributi normalizzati:

$$\sum_1^7 w_i * x_{normalizzato_i}$$

$$w_i = \begin{cases} 0.25, & \text{se } x_i = \text{topicuseful, elitedim} \\ 0.15, & \text{se } x_i = \text{compliment hot} \\ 0.1, & \text{se } x_i = \text{fans, compliment profile, useruseful} \\ 0.05, & \text{se } x_i = \text{compliment list} \end{cases}$$

Si ricavano infine i 10 utenti con score più alto.

In StackExchange i dati già sono divisi per topic. Si accede, quindi, con l'argomento in ingresso alla collezione corrispondente di MongoDB. Si ricavano i post relativi all'argomento, si filtrano solo quelli di risposta. Segue

un ulteriore filtraggio in base alle score associate al post: si mantengono tutti i post con score maggiore della media, disposti poi in ordine decrescente. Ancora una volta, si limitano i post ai migliori 100 per le considerazioni fatte precedentemente. Si passa di seguito alla ricerca degli esperti. Si accede alla collezione degli utenti e, mediante inner join, si ricavano gli utenti relativi ai migliori post trovati. Segue il raggruppamento dei post sugli utenti, per ciascuno dei quali viene calcolata la somma degli score dei suoi post relativi al topic di riferimento. Questo attributo viene combinato alla reputazione dell'utente per calcolare la valutazione di competenza. La reputazione dell'utente è un'informazione sintetica che calcola StackExchange in base agli apprezzamenti ed alla storia dell'utente. Viene sfruttata per determinare l'autorevolezza dell'opinione dell'utente. Si definisce nuovamente la metrica per la valutazione della competenza sfruttando la normalizzazione degli attributi e dando a ciascuno di essi un peso del 50%.

$$\sum_1^2 0.5 * x_{normalizzato_i}$$

```
# encoding=utf8
import sys
reload(sys)
sys.setdefaultencoding('utf8')
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, avg, length, min, max, sum
import sys

#Creazione della sessione Spark con relativa configurazione
#(wrapper di SparkContext)
spark = SparkSession.builder.appName("ExpertFinding")
    .config("spark.mongodb.input.uri",
            "mongodb://127.0.0.1/BDABI.yelpbusiness")
    .getOrCreate()

#Valutazione degli argomenti di input:
#      - nessun argomento di input —> Topic non selezionato
#      - almeno un argomento di input —> Creazione della stringa
#                                           corrispondente al topic da
#                                           cercare in MongoDB
if len(sys.argv)>1:
    topicstack = ""
    for i in range(1,len(sys.argv)):
```

```

    line = sys.argv[i].capitalize()      #argomento in input tutto
#                                         minuscolo con prima lettera maiuscola
    linestack = sys.argv[i].lower()
    if i==1:
        topic = line
    else:
        topic = topic+" "+line           #spazio tra piu' parole dello
#                                         topic
    topicstack = topicstack+linestack
print ("Topic"+selezionato+"%s") %topic

collectionpost = "stackexchange"+topicstack+"post"
collectionuser = "stackexchange"+topicstack+"user"
notyelp = False
notstack = False

#Normalizzazione in (0,1): (x-min)/(max-min)
def normalizza(df, cl):
    minimo = df.agg(min(cl)).collect()[0][0]
    massimo = df.agg(max(cl)).collect()[0][0]
    return((df[cl]-minimo)/(massimo-minimo))

#Lettura dalla collezione yelpbusiness di MongoDB
    business = spark.read.format("com.mongodb.spark.sql.DefaultSource")
        .load()

#Ricerca in business dei locali relativi al topic selezionato
    category = business.where(col('categories').like(topic+"%")
        | col('categories').like("%"+topic+"%")
        | col('categories').like("%"+topic)
        | col('categories').like(topic))
        .select("business_id")

#Lettura della collezione post di stackexchange
    post = spark.read.format("com.mongodb.spark.sql.DefaultSource")
        .option("uri", "mongodb://127.0.0.1/BDABI."+collectionpost)
        .load()

    if category.count()>0:
        print ("YELP")

```



```

#Lettura dalla collezione yelpreview di MongoDB
review = spark.read.format("com.mongodb.spark.sql.DefaultSource")
    .option("uri","mongodb://127.0.0.1/BDABI.yelpreview")
    .load()
#Selezione delle colonne di interesse di review
reviewsel = review.select('business_id', 'user_id', 'useful')
#Join per la ricerca delle recensioni appartenenti ai locali
#precedentemente trovati
reviewtopic = category.join(reviewsel,
    category.business_id==reviewsel.business_id, "inner")
    .select('user_id', 'useful')
#Filtraggio delle recensioni con utilita' inferiore alla media
#ed ordinamento discendente
reviewfilt = reviewtopic
    .filter(reviewtopic['useful'] >
        reviewtopic.agg(avg(col("useful"))).collect()[0][0])
    .orderBy('useful', ascending=False)
if reviewfilt.count() > 100:
    reviewfilt = reviewfilt.limit(100)           #riduzione a 100 righe
    # con maggiore useful

#Lettura della collezione yelpuser di MongoDB
user = spark.read.format("com.mongodb.spark.sql.DefaultSource")
    .option("uri","mongodb://127.0.0.1/BDABI.yelpuser")
    .load()
usersel = user
    .select('user_id', 'name', 'useful', 'fans', 'elite',
        'compliment_hot', 'compliment_profile',
        'compliment_list')
    .withColumnRenamed("useful", "useruseful")

#Join recensioni-utenti, calcolo della dimensione di elite,
#selezione colonne di interesse e somma di useful delle recensioni
#affendenti allo stesso utente
#La dimensione di elite e' la lunghezza della stringa
reviewer = reviewfilt.join(usersel,
    reviewfilt.user_id==usersel.user_id, "inner")
    .withColumn('elitedim', length('elite'))
    .groupBy('name', 'useruseful', 'fans', 'elitedim',
        'compliment_hot', 'compliment_profile',
        'compliment_list', usersel.user_id)

```

```

        .agg({ 'useful': 'sum' })
        .withColumnRenamed("sum(useful)", "topicuseful")

#Calcolo score di ogni utente
#Gli attributi contribuiscono con peso differente al calcolo
#dello score: 5% list, 10% fans, useruseful e profile, 15% hot,
#25% elite e topicuseful
    userscore = reviewer.withColumn('score',
        (0.25*normalizza(reviewer, 'topicuseful'))+
        (0.1*normalizza(reviewer, 'fans'))+
        (0.25*normalizza(reviewer, 'elitedim'))+
        (0.15*normalizza(reviewer, 'compliment_hot'))+
        (0.1*normalizza(reviewer, 'compliment_profile'))+
        (0.05*normalizza(reviewer, 'compliment_list'))+
        (0.1*normalizza(reviewer, 'useruseful')))
    .select('user_id', 'name', 'score')
    .orderBy('score', ascending=False)
    userscore.show(10)
else:
    notyelp = True

if post.count()>0:
    print("STACK_EXCHANGE")
    postsel = post.filter(post['_PostTypeId']=="2")
    .select("_postId", "_Score", "_OwnerUserId")
    .filter(post['_Score'] >
        post.agg(avg(col("_Score"))).collect()[0][0])
    .orderBy('_Score', ascending=False)

    if postsel.count()>100:
        postsel = postsel.limit(100)           #riduzione a 100 righe con
                                                maggiore score
#

#Lettura della collezione user di stackexchange
    user = spark.read.format("com.mongodb.spark.sql.DefaultSource")
    .option("uri", "mongodb://127.0.0.1/BDABI."+collectionuser)
    .load()
    usersel = user.select('_userId', '_DisplayName', '_Location',
        '_AboutMe', '_Reputation')

#Join post-utenti, selezione colonne di interesse e somma di

```

```

# score dei post afferenti allo stesso utente
expert = postsel.join(usersel,
                      postsel['_OwnerId']==usersel['_userId'], "inner")
    .groupBy(' _userId ', ' _DisplayName ', ' _Location ', ' _AboutMe ',
            ' _Reputation ')
    .agg({ ' _Score ': 'sum' })
    .withColumnRenamed("sum(_Score)", "topicscore")

#Calcolo dello score di ogni utente
expertscore = expert.withColumn('scoreexpertise',
                                (0.5*normalizza(expert, 'topicscore'))+
                                (0.5*normalizza(expert, ' _Reputation ')))
    .orderBy('scoreexpertise', ascending=False)
expertscore.show(10)
expertscore.select(' _DisplayName ', ' _AboutMe ').show(10, False)

else:
    notstack = True

if (notyelp and notstack):
    print("Topic□inesistente")

else:
    print("Topic□non□selezionato")

```

**Listing 2.4:** Expert Finding

Infine, per eseguire il codice Python, si sfrutta Spark con il package mongo-spark-connector. Per garantire l'esecuzione è necessario garantire sufficiente memoria per il driver, pena il fallimento dell'esecuzione. Sono stati dedicati al driver 2GB di memoria centrale. L'output è direzionato verso un file testo per distinguerlo dalle informazioni di controllo dell'esecuzione.

```

spark-submit
—packages org.mongodb.spark:mongo-spark-connector_2.12:2.4.0
—driver-memory 2g
/home/vmadmin/src/ExpertFinding.py topic >risultato.txt

```

**Listing 2.5:** Driver Memory

**Input:** topic

**Result:** experts

```

if topic composto da almeno una parola then
    Trasformazione topic in parole staccate, minuscole e con la prima
    lettera maiuscola per Yelp;
    Trasformazione topic in parole unite e minuscole per StackExchange;
    notYelp = False; notStack = False;
    Recupero da MongoDB le imprese di Yelp con categoria == topic;
    Recupero da MongoDB i post di StackExchange appartenenti alla
    collezione di topic;
    if numero imprese > 0 then
        Recupero da MongoDB le recensioni di Yelp e selezione degli
        attributi di interesse;
        Recupero delle recensioni relative alle imprese;
        Filtraggio recensioni con utilità > media(utilità);
        Riduzione ad almeno le 100 recensioni più utili;
        Recupero da MongoDB gli utenti di Yelp e selezione degli
        attributi di interesse;
        Recupero degli utenti autori delle recensioni più utili;
        Raggruppamento delle recensioni per utente e somma dell'utilità
        delle recensioni di uno stesso utente;
        Calcolo competenza come somma pesata degli attributi più
        importanti normalizzati;
        Ordinamento decrescente per competenza e selezione dei primi
        10 utenti;
    else
        notYelp = True
    if numero post > 0 then
        Filtraggio dei post di risposta e selezione degli attributi di
        interesse;
        Filtraggio dei post con score > media(score);
        Riduzione ad almeno i 100 post con score più alto;
        Recupero da MongoDB gli utenti di StackExchange appartenenti
        alla collezione di topic e selezione degli attributi di interesse;
        Recupero degli utenti autori dei migliori post;
        Raggruppamento dei post per utente e somma dello score dei
        post di uno stesso utente;
        Calcolo competenza come somma pesata degli attributi più
        importanti normalizzati;
        Ordinamento decrescente per competenza e selezione dei primi
        10 utenti;
    else
        notStack = True
    if notYelp AND notStack then
        print "Topic inesistente";
else
    print "Topic non selezionato";

```

**Algorithm 1:** Expert Finding Pseudocode

## Capitolo 3

# Risultati sperimentali

Attraverso i comandi del tipo in **Figura 2.1** sono stati caricati i file JSON nel database BDABI di MongoDB nella collezione *yelp*+nomefile. Attraverso il codice in **Figura 2.2** sono stati caricati i file XML nel database BDABI di MongoDB nella collezione *stackexchange*+topic+nomefile. Al seguito di questa fase di caricamento, rendiamo operativo MongoDB per le successive elaborazioni. In **Figura 3.1** osserviamo alcune delle centinaia di collezioni caricate.

```

> use BDABI
switched to db BDABI
> show collections
stackexchange3dprintingpost
stackexchange3dprintinguser
stackexchangeacademiapost
stackexchangeacademiauser
stackexchangeaiuser
stackexchangeaiuser
stackexchangeandroidpost
stackexchangeandroiduser
stackexchangeanimepost
stackexchangeanimeuser
stackexchangeapplepost
stackexchangeappleuser
stackexchangearduinopost
stackexchangearduinouser
stackexchangeaskubuntuuser
stackexchangeaskubuntuuser
stackexchangeastronomyuser
stackexchangeastronomyuser
stackexchangeaugurpost
stackexchangeauguruser
stackexchangeaviationpost
stackexchangeaviationuser
stackexchangeavppost
stackexchangeavppuser
stackexchangebeerpost
stackexchangebeeruser
stackexchangebicyclespost
stackexchangebicyclesuser
stackexchangebioinformaticsuser
stackexchangebioinformaticsuser
stackexchangebiologypost
stackexchangebiologyuser
stackexchangebitcoinpost
stackexchangebitcoinuser
stackexchangetravelpost
stackexchangetraveluser
stackexchangetridionpost
stackexchangetridionuser
stackexchangeukrainianpost
stackexchangeukrainianuser
stackexchangeunixpost
stackexchangeunixuser
stackexchangeuserexperiencepost
stackexchangeuserexperienceuser
stackexchangevegetarianismpost
stackexchangevegetarianismuser
stackexchangevipost
stackexchangevipuser
stackexchangewebappsuser
stackexchangewebmasterspost
stackexchangewebmastersuser
stackexchangewindowsphonepost
stackexchangewindowsphoneuser
stackexchangewoodworkingpost
stackexchangewoodworkinguser
stackexchangewordpresspost
stackexchangewordpressuser
stackexchangeworkspacepost
stackexchangeworkspaceuser
stackexchangeworldbuildingpost
stackexchangeworldbuildinguser
stackexchangewriterspost
stackexchangewritersuser
yelpbusiness
yelpcheckin
yelpphoto
yelppreview
yelptip
yelpuser

```

Figura 3.1: MongoDB

Conclusasi questa fase, non resta che sfruttare i dati che si dispone per effettuare la ricerca degli esperti su di un determinato argomento. Ricerchiamo gli esperti di *Golf*. Lanciamo il comando in **Figura 2.5** con *golf* come topic. Osserviamo un tempo di elaborazione di poco più di 12 minuti. Essa ha coinvolto principalmente il dataset di Yelp, grazie alle recensioni sui campi da golf. Si ottengono gli esperti relativi.

```

Topic selezionato: Golf
YELP
+-----+-----+-----+
| user_id | name | score |
+-----+-----+-----+
| --2vR0DIsmQ6WfcSz... | Harald | 0.6128201926952267 |
| Z88N6qly3Dp5C06Xb... | Janice | 0.4816168137363288 |
| nkN_do3fJ9xekchVC... | Jeremy | 0.37949707107989017 |
| AbMjnK0wg736fcIu8... | Michael | 0.36829495634104636 |
| fgwI3rYH0vlipfVfC... | Emi | 0.34889763463524603 |
| rMsB82tk9u0B6JumP... | Brett | 0.31008172530736716 |
| 3mNz5nQFTIBQm0oU5... | Michelangelo | 0.3087006619216034 |
| 13f_vtUZEmIzweL91... | Michelle | 0.2975279223565622 |
| aIbYx0V_3dBIPUcnl... | Tess | 0.2911744071500844 |
| zFYs8gSUYDvXkb607... | Joyce | 0.29038847104296245 |
+-----+-----+-----+
only showing top 10 rows

```

Figura 3.2: Esperti di golf

Analogamente effettuiamo un'ulteriore ricerca di esperti di animali. Osserviamo che il tempo di esecuzione è poco più di 12 minuti. La ricerca è stata effettuata su entrambi i dataset. Risulta interessante osservare le descrizioni che hanno dato gli utenti di loro stessi. La maggior parte di esse fanno riferimento ad un aspetto della vita dell'utente correlato agli animali. Sono dunque persone che reputano il loro rapporto con gli animali come una delle cose primarie da inserire in una descrizione personale.

**Zaralynda** Avid reader, feminist, owned by cats, sometime gamer, and quilter. Currently share my home with 3 cats.

**Yvette Colomb** Animal lover and animal rights activist.

**Trond Hansen** I am 53 years old (young). My interests are very broad: science, astronomy, biology, geology, marine biology. I simply want to know how every thing works, I have a cat named Trine. I have always had a cat.

**Rebecca RVT** Graduated as Veterinary Technician in spring 2013, got certified in summer of 2013. Currently work primarily with animal (cats and dogs). I have 11 years experience with exotic pets (pocket pets, reptiles and birds) with 2 years of working at an exotic pet practice.

**James Jenkins** I volunteer with <http://www.petfinder.com/shelters/PA323.html> Rabbit Wranglers. I can often be found at local community events walking a rabbit on a leash, providing education and offering tips for adopting and rating a pet house rabbit into the family.

Topic selezionato: Pets  
YELP

user_id	name	score
Tqm7Wu7IBJ1td3Ab5...	Brian	0.44115631119814847
M9rRM6Eo5YbKLMG5...	Aileen	0.4134393090710687
wTfb2nfzPIyFcYQAr...	PrincessCandyEmpire	0.4047377559487504
zFYs8gSUYDvXkb607...	Joyce	0.3929949240832057
A0j21z2Q1HGic7jW6...	Georgie	0.33843725767560195
DK57YibC5ShBmqQl9...	Karen	0.3240832341973503
Fv0e9RIV9jw5TX3ct...	Christie	0.3112994076757475
rKDLq635fyrnVzg4G...	Maila	0.2955102518043695
xsMd60nEJ6_Np6Es0...	Gerard	0.25888538160993735
Jtq_pKd7GVbXvFY8Y...	Marissa	0.25716393263117915

only showing top 10 rows

STACK EXCHANGE

_userId	_DisplayName	_Location	_AboutMe	_Reputation	topicscore	scoreexpertise
224	Zaralynda	Interwebs	<p>Avid reader, f...	15990	231	0.9380414733664971
32	John Cavan	Toronto, Canada	<p>I tried subtle...	18207	160	0.8317535545023697
6796	Yvette Colomb	null	<p>Animal lover a...	13645	196	0.7895673405119628
13	James Jenkins	Pittsburgh, PA	<p>I volunteer wi...	18177	91	0.6674080351236993
7526	Rebecca RVT	Canada	<p>Graduated as V...	9872	105	0.46848345205736364
57	Beofett	Pennsylvania	<p>This page inte...	6354	144	0.46258292381909305
7612	trond hansen	Røyken, Norway	<p>i am 53 years ...	7112	122	0.4316340578452827
7852	motosubatsu	null	<p>This section l...	4299	154	0.4288484956692727
481	Spidercat	Wherever I feel like	<p>Just your frie...	11109	69	0.4177458496037485
13155	Alan T.	Ottawa, ON, Canada	<p>Administrative...	1458	176	0.40158373467980535

only showing top 10 rows

Figura 3.3: Esperti di animali

```

|Zaralynda |<p>Avid reader, feminist, owned by cats, sometime gamer, and quilter.</p>
<p>Currently share my home with 3 cats.</p>
<p></p>
|Yvette Colomb|<p>Animal lover and animal rights activist.</p>

|James Jenkins|<p>I volunteer with <a href="http://www.petfinder.com/shelters/PA323.html" rel="nofollow noreferrer">Rabbit Wranglers</a>.
I can often be found at local community events walking a rabbit on a leash, providing education and offering tips for adopting and integ
rating a pet house rabbit into the family</p>

|Rebecca RVT |<p>Graduated as Veterinary Technician in spring 2013, got certified in summer of 2013. Currently work primarily with small
animal (cats &amp; dogs).</p>
<p>I have 11 years experience with exotic pets (pocket pets, reptiles and birds) with 2 years of working at an exotic pet practice.</p>

|trond hansen |<p>i am 53 years old(young)my interests is very broad,sience-astronomy-biology-geology-marine biology.i simply want to kno
w how every thing works,i have a cat named trine i have allways had a cat.</p>

```

Figura 3.4: About Me degli esperti di animali

Infine ricerchiamo gli esperti di informatica. L'esecuzione impiega 30 secondi e coinvolge solo il dataset di StackExchange. Osserviamo ancora una volta come le descrizioni degli utenti siano coerenti con la tipologia di esperto



cercata: un ricercatore del Technion, un ingegnere del software di Google, un dottorando dell'Università di British Columbia, un professore dell'Università dell'Illinois.

**Yuval Filmus** Assistant Professor in the Department of Computer Science at the Technion.

**Raphael** I am a computer scientist by training, which means I now think like one: always analysing, abstracting, reducing, problem solving. In addition, I picked up some affection and, hopefully, ability for actually building software over the years. You can take a look over on <https://github.com/reitzig> Github. During my time at university I have found a passion for teaching, by which I mean helping people learn. Some say I was quite the nitpicker; it's for your best, I promise! In my free time I play games, read books, code, work out, enjoy music, and roam the webs.

**Kaveh** <http://ca.linkedin.com/in/ghasemloo> Software Engineer at <http://www.google.com> Google. Ph.D. in Computer Science, <https://www.utoronto.ca/> University of Toronto, <http://web.cs.toronto.edu/> Department of Computer Science, <http://www.cs.toronto.edu/theory/index.php> Theory Group. Thesis: <http://www.cs.toronto.edu/~kaveh/papers/phd-thesis.pdf> "Uniformity and Nonuniformity in Proof Complexity", 2016. Ex-moderator on <http://cstheory.stackexchange.com> cstheory.

**Jmite** I am Joey Eremondi, a PhD Student at the <https://www.cs.ubc.ca/> University of British Columbia. I do research in Programming Languages and Theory of Computation, particularly with dependent types. My <http://dspace.library.uu.nl/handle/1874/337692> Masters Thesis was on improving error messages for higher order unification. I've also co-authored a few papers on reversal-bounded counter automata. I have an M.Sc in Computing Science from <http://www.cs.uu.nl/> Utrecht University, a B.Sc. Honours in Computer Science, and a B.Sc. 4-year in Mathematics, both from the <http://www.usask.ca/> "rel= University of Saskatchewan.

**JeffE** I am a full professor of <http://www.cs.uiuc.edu> computer science at the University of Illinois, Urbana-Champaign. I teach <http://www.cs.uiuc.edu/~jeffe/teaching/algorithms> algorithms.

Topic selezionato: Computer Science  
STACK EXCHANGE

_userId	_DisplayName	_Location	_AboutMe	_Reputation	topicscore	scoreexpertise
683	Yuval Filmus	Haifa, Israel	<p>Assistant Prof...	199861	541	0.7592397043294614
9550	David Richerby	UK		72177	997	0.679749184850765
755	D.W.	null	null	104210	618	0.5599872049843089
98	Raphael	Nürnberg, Germany				
<p>I am a compu...	58499	462	0.36297172780914955			
41	Kaveh	Toronto, Canada	<p><a href="http:...	17993	633	0.3516616543364668
2253	jmite	University of Bri...	<p>I am Joey Erem...	23069	579	0.33588194366644075
39	Gilles	null	<p>Moderator on ...	33906	277	0.20361188381776146
133	Sebastian	null		3516	293	0.13583688963789126
22096	tsleyson	United States	null	2763	258	0.11546884290522869
72	JeffE	Urbana, IL	<p>I am a full pr...	7801	208	0.10170575489665595

only showing top 10 rows

Figura 3.5: Esperti di informatica

La metrica di valutazione della competenza riesce efficacemente ad individuare gli esperti nel settore. Le differenze sostanziali nei tempi di elaborazione sono dovute alla composizione dei dataset. Essendo StackExchange già suddiviso per topic, presenterà un numero di tuple nettamente inferiore alle centinaia di migliaia di Yelp, che mantiene i dati di tutta la piattaforma senza scrematura. La composizione di Yelp necessita inoltre di un'operazione di Join aggiuntiva per ricavare i topic dalle aziende, e di conseguenza le recensioni relative. Differenti configurazioni tecniche di Spark, variando il numero di esecutori, di core per ciascuno di essi, e di memoria loro disponibile, possono portare ad una riduzione dei tempi globali di al più 10 secondi, suggerendo che non siano le limitazioni tecniche della piattaforma a vincolare eccessivamente i tempi.