



UNIVERSITÀ DELLA CALABRIA

DIPARTIMENTO DI  
INGEGNERIA INFORMATICA,  
MODELLISTICA, ELETTRONICA  
E SISTEMISTICA

DIMES

Modelli e Tecniche per Big Data

# Progetto Flickr

Anno Accademico 2021/2022

Viviana Colantonio  
Matricola 224473

Cristian Giuseppe Costa  
Matricola 227507

# Indice

<b>1 Introduzione e Analisi Dataset</b>	<b>2</b>
1.1 Struttura del dataset . . . . .	2
1.2 Preprocessing del dataset . . . . .	5
1.3 Sommario delle analisi qualitative effettuate . . . . .	6
<b>2 Alcune query su post e utenti</b>	<b>7</b>
2.1 Analisi dell'utilizzo del social da parte degli utenti . . . . .	7
2.2 Analisi riguardanti l'utilizzo dei tag . . . . .	8
2.3 UDF . . . . .	9
<b>3 Trajectory Mining</b>	<b>12</b>
3.1 Prima analisi del problema . . . . .	12
3.2 Approccio adottato . . . . .	12
3.2.1 Task 1 : Individuazione di post credibili . . . . .	13
3.2.2 Task 2 : Attribuzione di un luogo ad un post . . . . .	13
3.2.3 Task 3 : Trasformazione del dataset . . . . .	19
3.2.4 Task 4 : Analisi e confronto dei risultati ottenuti . . . . .	20
<b>4 User Clustering</b>	<b>22</b>
4.1 Task 1 : User Embedding . . . . .	22
4.2 Task 2 e 3: Clustering e selezione parametri . . . . .	25
4.3 Task 4 : Interpretazione dei risultati . . . . .	26
4.4 Ulteriori elementi di categorizzazione . . . . .	30
<b>5 Analisi delle prestazioni su dati sintetici</b>	<b>33</b>

# Capitolo 1

## Introduzione e Analisi Dataset

Per il progetto assegnatoci si è scelto di utilizzare il dataset presente nella cartella flickr1x, in quanto i dati contenuti in tutti gli altri dataset forniti risultano essere generati sinteticamente. Quindi, gli altri file sono stati utilizzati soltanto per generare analisi prestazionali (vedi Capitolo 5).

Si è optato per Spark + Scala.

### 1.1 Struttura del dataset

Il dataset, memorizzato in formato JSON, è stato ottenuto sfruttando le API Flickr, il social network dedicato alla condivisione di foto e video, lanciato nel 2004 dalla società canadese Ludicorp e che ad oggi vanta milioni di utenti. Si riportano di seguito le denominazioni dei campi presenti nel dataset (in particolare di un singolo oggetto JSON di esempio) ed il loro significato, ottenuto dalle specifiche delle API Flickr<sup>1</sup>.

```
1  {
2      "server": "173",
3      "notes": [],
4      "publicFlag": true,
5      "placeId": "FphPyURWU7ux6h4",
6      "description": "St Peter's church in Rome",
7      "secret": "b96b1fece0",
8      "originalFormat": "jpg",
9      "media": "photo",
10     "title": "Basilica di San Pietro",
```

---

<sup>1</sup>Servizi Flickr (2022). URL: <https://www.flickr.com/services/api/> (visitato il 07/02/2022).

```
11      "iconServer": "",  
12      "urls": [],  
13      "farm": "1",  
14      "id": "430793876",  
15      "hasPeople": false,  
16      "datePosted": "Mar 22, 2007 11:58:08 PM",  
17      "views": 60,  
18      "originalSecret": "",  
19      "owner": {  
20          "photosCount": 0,  
21          "admin": false,  
22          "revFamily": false,  
23          "pro": false,  
24          "iconServer": 0,  
25          "iconFarm": 0,  
26          "revContact": false,  
27          "filesizeMax": 0,  
28          "bandwidthUsed": 0,  
29          "bandwidthMax": 0,  
30          "id": "91071733@N00",  
31          "revFriend": false,  
32          "username": "swashford"  
33      },  
34      "comments": 0,  
35      "originalHeight": 0,  
36      "familyFlag": false,  
37      "rotation": -1,  
38      "mediaStatus": "ready",  
39      "geoData": {  
40          "latitude": 41.90245,  
41          "accuracy": 16,  
42          "longitude": 12.456661  
43      },  
44      "friendFlag": false,  
45      "url": "https://flickr.com/photos/91071733@N00/430793876",  
46      "originalWidth": 0,  
47      "tags": [  
48          {  
49              "count": 0,  
50              "value": "holiday"  
51          },  
52          {  
53              "count": 0,  
54              "value": "vatican"  
55          },  
56          {  
57              "count": 0,  
58              "value": "stpeters"  
59          },  
60          {
```

```

61      "count": 0,
62      "value": "rome"
63    }
64  ],
65  "license": "",
66  "iconFarm": "",
67  "lastUpdate": "Dec 10, 2014 1:09:09 AM",
68  "favorite": false,
69  "dateTaken": "Jan 1, 0001 12:00:00 AM",
70  "primary": false,
71  "pathAlias": ""
72 }

```

- `server`, `iconServer`, `farm`, `secret`, `urls`, `originalSecret` sono utilizzati per la costruzione dell'url identificativo del post
- `notes` identifica una lista di eventuali annotazioni, ma nell'intero dataset risultano essere tutte vuote, e quindi non significative
- `publicFlag` è un booleano posto a `true` solo se si tratta di un post di un utente con profilo pubblico
- `placeId` è l'identificativo delle coordinate geografiche del post
- `description` indica la descrizione del post inserita dall'autore dello stesso
- `originalFormat` indica l'estensione del contenuto multimediale caricato a seguito della pubblicazione del post
- `media` dà indicazioni del tipo di contenuto multimediale caricato all'interno del post
- `title` è un campo testuale che specifica il titolo del post dato dall'utente
- `id` corrisponde all'identificativo univoco del post
- `datePosted` indica la data di caricamento del post all'interno della piattaforma
- `views` corrisponde al numero di visualizzazioni del post fino al momento del download dei suoi metadata
- `owner` contiene al suo interno dei campi che identificano il creatore del post, tra i quali:
  - `revFamily`, `revCount` e `revContact` si riferiscono alla relazione tra il creatore del post e l'utente che ha utilizzato le API (se non autenticato non sono presenti)

- `pro` è un booleano posto a `true` se il creatore del post dispone di un piano di Flickr a pagamento
- `id` è l'identificatore univoco dell'utente
- `username` definisce il nickname dell'utente, anch'esso univoco.
- `comments` è un contatore per il numero di commenti relativi al post
- `geoData` definisce i campi descrittivi delle coordinate GPS del post (se geotaggato), considerando latitudine, longitudine e accuratezza della misurazione.
- `url` definisce l'url completo del post
- `tags` definisce i campi descrittivi dei tag inseriti nel post. Tra questi:
  - `count`, un contatore di utilizzo del tag
  - `value`, ovvero la stringa identificativa del tag
- `favorite` è un booleano posto a `true` solo nel caso in cui l'utente che ha sfruttato le API abbia aggiunto tra i suoi post preferiti proprio quello di cui ha scaricato i metadata
- `dateTaken` definisce invece la data in cui è stato catturato il contenuto multimediale pubblicato nel post

Di tutti gli altri campi che non sono stati menzionati nella precedente descrizione non si è riusciti a trovare la documentazione, e pertanto sono stati ignorati.

## 1.2 Preprocessing del dataset

Prima di procedere con la realizzazione della analisi qualitative del dataset, è stata effettuata una fase di preprocessing e di pulizia dello stesso, in modo da poter rimuovere campi ritenuti poco rilevanti e da poter alleggerire poi il lavoro svolto dalle query. In particolare, tutto ciò è stato realizzato all'interno del metodo di caricamento del dataset, che quindi si occupa, sfruttando Spark SQL, di leggere il file JSON e restituire un Dataset Spark contenente oggetti `FlickrPost`. Come si può notare dallo snippet seguente, all'interno dell'oggetto sono stati mantenuti soltanto i campi considerati rilevanti ai fini della analisi.

```

1  case class FlickrPost(description: String, title: String, urls:
2    ↪ List[String], id: String, datePosted: Instant, views: Long, owner:
3    ↪ FlickrPostOwner, comments: Long, originalHeight: Long, familyFlag:
4    ↪ Boolean, geoData: FlickrGeoData, url: String, tags: List[FlickrPostTag],
5    ↪ lastUpdate: Instant, dateTaken: Instant) {
6
7  }
```

Inoltre, ogni oggetto `FlickrPost` innesta al suo interno un oggetto di tipo `FlickrPostOwner`, `FlickrGeoData` ed una lista di oggetti `FlickrPostTag`, anch'essi definiti considerando solo i campi ritenuti utili.

### 1.3 Sommario delle analisi qualitative effettuate

Dopo aver effettuato uno studio del dataset di interesse, si è deciso di definire analisi qualitative che possono essere così categorizzate:

- analisi relative all'utilizzo del social da parte degli utenti, contenenti
  - query che realizzano una classificazione degli utilizzatori sulla base di numero di visualizzazioni totalizzate, numero di post pubblicati o entrambi i parametri
  - query che definiscono serie temporali riguardanti l'andamento del numero di pubblicazioni effettuate durante l'anno, durante un particolare mese dell'anno, oppure giornalmente.
- analisi relative all'utilizzo dei tag all'interno dei post, contenenti
  - query che definiscono serie temporali riguardanti l'andamento del numero di post contenenti un determinato tag durante un particolare anno, mese, oppure giornalmente
  - query che definiscono i tag più utilizzati sulla base di visualizzazioni raccolte dal totale dei post contenenti quei tag
- analisi di trajectory mining sulla base dei *geotag* dei post, in modo da definire i percorsi ricorrenti degli utenti
- analisi di clusterizzazione degli utenti, sulla base dei titoli e delle descrizioni inserite nei post da loro pubblicati
- analisi che sfruttano concetti di machine learning, in modo da riuscire a categorizzare le immagini poste dagli utenti sulla base del loro contenuto, e a definire la lingua utilizzata dagli utenti per scrivere i loro post.

## Capitolo 2

# Alcune query su post e utenti

Nel capitolo seguente verrà descritto il funzionamento delle query relative all'utilizzo del social da parte degli utenti e quelle riguardanti l'utilizzo dei tag all'interno dei post presenti nel dataset di interesse. In particolare, per ognuna di esse verrà riportata la signature del metodo relativo e si descriverà il processo utilizzato per realizzarle.

### 2.1 Analisi dell'utilizzo del social da parte degli utenti

Per stilare una classifica degli utenti maggiormente influenti all'interno del social network, sono state utilizzate tre query accomunate dagli stessi parametri in input e dallo stesso valore di ritorno, delle quali si riporta la signature nello snippet seguente.

```
1 def mostInfluential(dataset: Dataset[FlickrPost]): Dataset[Row] = {...}
```

Una volta ottenuto il dataset di FlickrPost, questo viene mappato considerando soltanto le informazioni relative all'autore del post (ovvero il suo id ed username) e lo score ottenuto. Quest'ultimo può tener conto sia del numero di visualizzazioni totalizzate che del numero di post pubblicati (considerando  $score = \frac{views}{count} + \log(count)$ ), oppure soltanto di uno dei precedenti parametri. In figura 2.1 si riporta un esempio del risultato ottenuto considerando lo score che tiene conto sia delle visualizzazioni che del numero di post.

Per analizzare l'andamento temporale delle pubblicazioni di ogni utente, invece, sono state realizzate delle serie temporali tramite query definite dalla seguente signature:

```
1 def trendOfPostForUserByYear(dataset: Dataset[FlickrPost], user: String):  
    Dataset[Row] = {...}
```

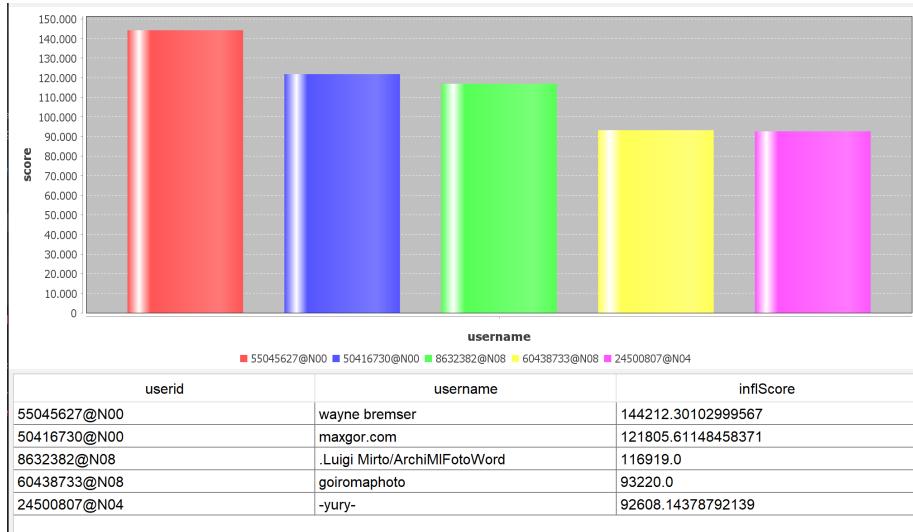


Figura 2.1: Esempio classifica utenti

Dopo aver caricato il dataset di partenza, questo viene filtrato sulla base dell’username passato in input, in modo da avere soltanto i post relativi a quell’utente. Successivamente, il risultato del seguente filtraggio viene messo in join con un dataset contenente il numero di post pubblicati dall’utente in una precisa data, oppure il numero di visualizzazioni ottenute. Infine si applicano delle funzioni di aggregazione (sum) in modo da ottenere il totale di visualizzazioni o post pubblicati per ogni data. Un esempio è mostrato in figura 2.2.

## 2.2 Analisi riguardanti l’utilizzo dei tag

In maniera molto simile a quanto discusso per i post nel paragrafo precedente, sono state definite delle analisi riguardanti l’utilizzo dei tag all’interno dei post contenuti nel dataset. La prima query si pone come obiettivo quello di definire i tag maggiormente utilizzati, sulla base del numero di post che li contengono, ed il totale delle visualizzazioni ottenuti dagli stessi. Si riporta di seguito la signature.

```
1 def mostUsedTags(dataset: Dataset[FlickrPost]): Dataset[Row] = {...}
```

Per definire questa analisi, una volta ottenuto il dataset di partenza, attraverso operazioni di mapping e reduce è stato realizzato un Dataset contenente una colonna per la stringa che definisce il tag, una per il totale di visualizzazioni, e un’ultima colonna per contenere il totale dei post nei quali compare il tag, per poi ordinarlo in maniera decrescente sulla base di entrambi i parametri. Si aggregano in OTHER i tag che non coprono una percentuale di post (ossia

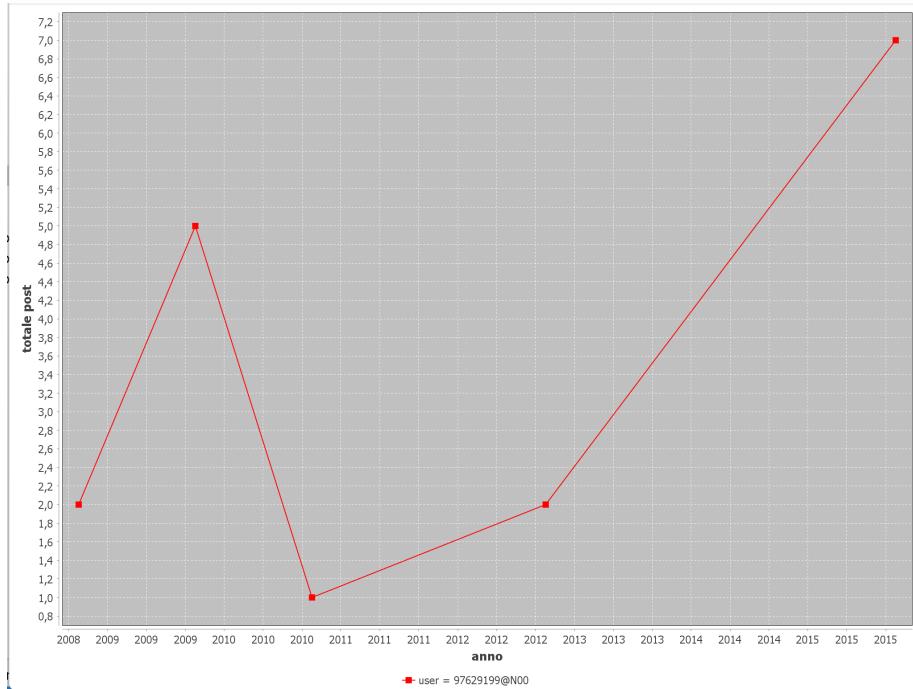


Figura 2.2: Esempio andamento post utente, per mese

un supporto) sufficientemente elevata (pari allo 0.5%, nel caso particolare). In figura 2.3 si riporta un esempio di grafico a torta ottenuto.

Le query che definiscono l’andamento temporale dell’utilizzo di un tag nel tempo sono accomunate dalla seguente signature:

```
1 def trendTagByDay(dataset: Dataset[FlickrPost], tag: String): Dataset[Row] =
  ↳ {...}
```

Il dataset principale viene filtrato sulla base del tag passato in input, in modo da ottenere soltanto i post che lo contengono, per poi mapparlo considerando soltanto l’id del post, la data in cui è stato postato e i tag utilizzati. Questo risultato intermedio è stato poi raggruppato sulla base della date, in modo da poter effettuare il conteggio dei post pubblicati. In figura 2.4 si riporta un esempio della serie temporale ottenuta considerando il tag “rome” e comparando i risultati per mese.

La stessa query è stata definita considerando una specifica data, oppure un particolare anno.

## 2.3 UDF

Per definire le analisi descritte nei paragrafi 2.1 e 2.2 sono risultate di particolare importanza le UDF per la trasformazione delle date in cui i post sono stati

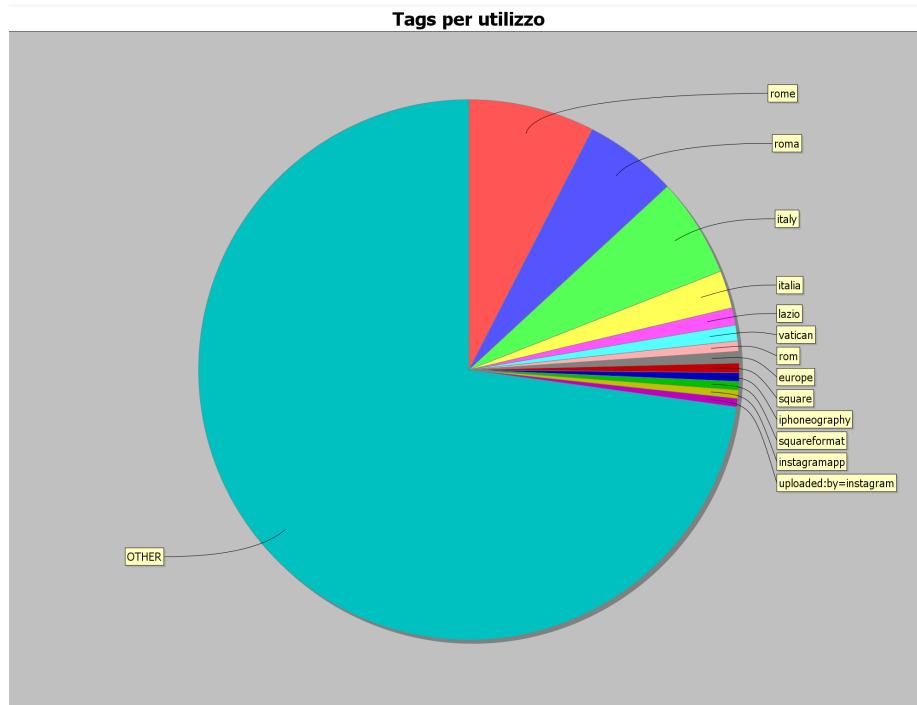


Figura 2.3: Esempio distribuzione dei tag

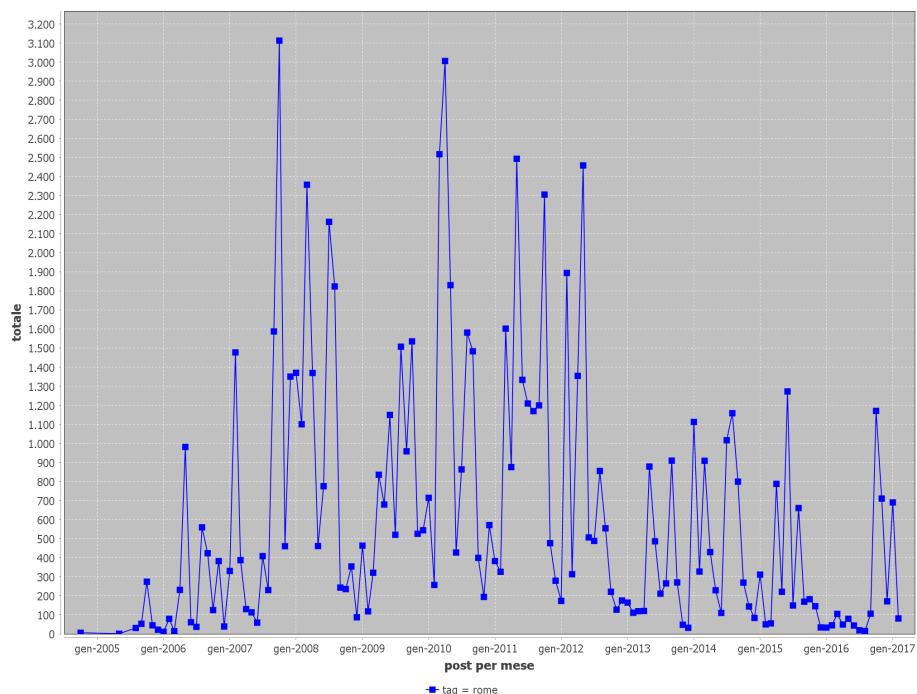


Figura 2.4: Esempio andamento tag, per mese

pubblicati, in modo da effettuare la trasformazione da stringa ad oggetto data.

```
1  @transient
2  val formatter = DateTimeFormatter.ofPattern("MMM d, yyyy h:mm:ss a")
3      .withLocale(Locale.ENGLISH)
4  @transient
5  val to_date_en = udf((x: String) => LocalDateTime.parse(x, formatter)
6      .toInstant(ZoneOffset.UTC))
7  @transient
8  val to_date_en_year = udf((x: String) => x.split("-")(0))
9  @transient
10 val to_date_en_month = udf((x: String) => x.split("-")(0) + " - " +
    → x.split("-")(1)) //anno-mese
```

Come si può notare dallo snippet, se per la trasformazione da stringa a data completa è stato possibile sfruttare la classe Java `java.time.format.DateTimeFormatter`, per le trasformazioni in anno oppure mese il formatter non è risultato funzionante e si è optato per una semplice trasformazione che lavora sulla struttura della stringa.

## Capitolo 3

# Trajectory Mining

### 3.1 Prima analisi del problema

Il dataset Flickr è stato oggetto di studi accademici nell'ambito del Social Media Mining, in particolare in [Belcastro et al. 2019] - paper il quale è in realtà incentrato sulla definizione di un framework per la definizione e computazione di attività di Social Media Mining - si definisce, si svolge e si mostrano i risultati di un'analisi di Trajectory Mining sul dataset in analisi.

La tecnica, definita *sequential pattern analysis*, si basa su **PrefixSpan** [Pei et al. 2004], un algoritmo in grado di inferire *frequent sequential patterns*, ossia pattern sequenziali *frequenti*, dove la nozione di frequenza è identica al concetto di frequenza nelle analisi di itemset frequenti, ossia si basa sulla nozione di supporto.

Attraverso **PrefixSpan**, è possibile effettuare *mining* di traiettorie più o meno popolari tra gli utenti di Flickr presenti nel dataset. Ovviamente ciò è possibile solo su post geotaggati e con date di acquisizione delle foto corrette.

Spark MLlib mette a disposizione una implementazione parallela dell'algoritmo PrefixSpan<sup>1</sup>, utilizzata ai fini di inferire sulla base dei post del dataset i percorsi di maggiore interesse.

### 3.2 Approccio adottato

Vi sono tuttavia alcune problematiche da risolvere, tra cui:

- **Task 1:** l'individuazione di post credibili rispetto all'analisi di Trajectory Mining

---

<sup>1</sup> Frequent Pattern Mining - Spark 3.2.1 Documentation (2022). URL: <https://spark.apache.org/docs/latest/ml-frequent-pattern-mining.html#prefixspan> (visitato il 04/02/2022).

- **Task 2** : l'attribuzione di un luogo ad un post (latitudine e longitudine sono troppo precise)
- **Task 3** : la trasformazione del dataset in uno compatibile con l'implementazione di PrefixSpan
- **Task 4** : l'interpretazione dei risultati.

### 3.2.1 Task 1 : Individuazione di post credibili

Il task 1 è risolto nel seguente modo: si eliminano dal dataset tutti i post per i quali

- la data di acquisizione della foto è mancante
- la data di acquisizione della foto è antecedente il 1/1/2004<sup>2</sup>
- la data di acquisizione della foto è conseguente al 1/1/2020.

Ciò è sintetizzato nel seguente snippet:

```

1  private def preprocess_for_trajectory_mining(dataset: Dataset[FlickrPost]): 
2    Dataset[FlickrPost] = {
3      dataset
4        .filter(_.dateTaken != null)
5        .filter(_.dateTaken.isAfter(LocalDateTime.of(2004, 1, 1, 0,
6          0).toInstant(ZoneOffset.UTC)))
7        .filter(_.dateTaken.isBefore(LocalDateTime.of(2020, 1, 1, 0,
8          0).toInstant(ZoneOffset.UTC)))
9        .filter(x => x.geoData != null && x.geoData.latitude != null &&
10           x.geoData.longitude != null)
11    }

```

### 3.2.2 Task 2 : Attribuzione di un luogo ad un post

Il task 2 è il più problematico: si potrebbe pensare di fare utilizzo dell'attributo `placeId` di Flickr, il quale dovrebbe rappresentare un cluster rispetto alle coordinate geografiche dei post; tuttavia esso non è sufficientemente informativo, e anzi risulta troppo impreciso, come è possibile osservare in 3.1.

Si è deciso pertanto di ricorrere ad un'altra soluzione.

In prima analisi, si è pensato di far uso di tecniche di clustering quali KMeans o DBScan. Tuttavia ciò richiedeva un *fine tuning* dei parametri non banale rispetto al task di interesse. In secondo luogo, si è pensato di definire staticamente

---

<sup>2</sup>Simbolicamente preso come istante zero, tenuto anche conto della data di creazione di Flickr

alcuni luoghi di interesse sulla base di coordinate di un centro e raggio di estensione, in parte sulla scia di quanto tracciato in [Belcastro et al. 2019], anche se in maniera estremamente semplificata.

Si è optato infine per un'opzione più generica basata su OpenStreetMap, in grado di fornire, potenzialmente, risultati di maggior interesse.

L'idea è la seguente:

1. si individuano le coppie (latitudine, longitudine) presenti nel dataset
2. si effettua un *rounding* a 3 cifre delle coppie (latitudine, longitudine) tenendo a mente la precisione (in metri) ottenibile con 3 cifre<sup>3</sup>
3. si filtrano le sole coppie distinte
4. si effettua una richiesta di tipo GET alle API Nominatim<sup>4</sup> di OpenStreetMap su tutte le coppie (latitudine, longitudine). Ogni richiesta fornirà un oggetto di tipo `GeoDItem`, contenente informazioni quali categoria del luogo, importanza e altre informazioni (vedi 4.4)
5. gli oggetti `GeoDItem` ottenuti al passo (4) saranno raccolti in un dataset - al quale ci si riferirà in seguito con il nome `datasetGeo`. Esso sarà propriamente filtrato sulla base di analisi delle distribuzioni degli attributi delle sue tuple e di rilevanza degli stessi
6. ad ogni post presente nel dataset, verranno assegnati tutti i luoghi così trovati (mediante join) filtrando solo quelli che si trovano nel raggio di 300 metri dalla posizione dello scatto fotografico
7. per ogni post, infine, si manterrà solo il luogo più *rilevante*, dove per rilevante si intende che occorre maggiormente nel dataset ottenuto al passo (5), in modo da garantire che
  - (a) ogni post che aveva un luogo al passo (5) lo avrà anche al passo (6)
  - (b) i luoghi “troppo precisi” vengano sostituiti da luoghi “più imprecisi” ma comunque nel raggio di 300 metri dallo scatto fotografico<sup>5</sup>

---

<sup>3</sup> *Measuring accuracy of latitude and longitude?* (2022). Geographic Information Systems Stack Exchange. URL: <https://gis.stackexchange.com/questions/8650/measuring-accuracy-of-latitude-and-longitude> (visitato il 04/02/2022).

<sup>4</sup> *Overview - Nominatim Documentation* (2022). URL: <https://nominatim.org/release-docs/develop/api/Overview/> (visitato il 04/02/2022).

<sup>5</sup> La basilica di San Pietro è lunga 218 metri, ad esempio

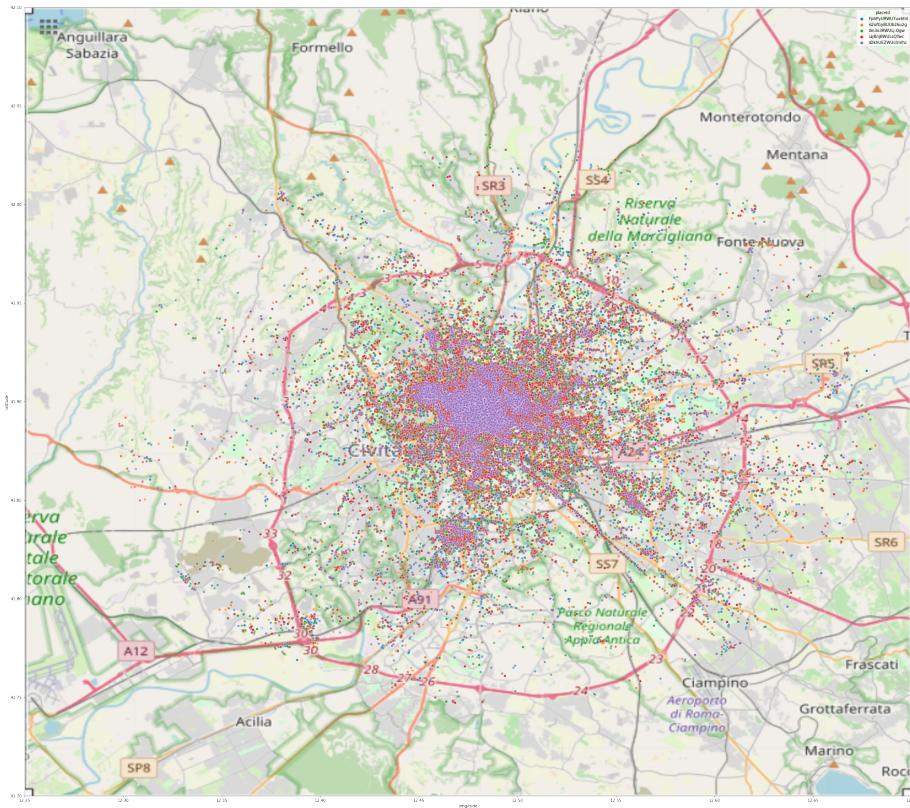


Figura 3.1: placeId, scatterplot con colori diversi rispetto ad alcuni cluster di Flickr

Un oggetto `GeoDFItem` è ha la seguente definizione:

```

1  case class GeoDFItem(importance: Double, category: String, address:
2    ↪ GeoDFItemAddress, typology: String, lat: Double, lon: Double)
3
4  {...}
5
6  case class GeoDFItemAddress(aeroway: String, amenity: String, building:
7    ↪ String, city: String,
8      club: String, commercial: String, country:
9        ↪ String, country_code: String,
10       county: String, craft: String, hamlet: String,
11         ↪ highway: String,
12        historic: String, house_number: String,
13          ↪ industrial: String, leisure: String,
14         locality: String, man_made: String, military:
15           ↪ String,
16          municipality: String, natural: String,
17            ↪ neighbourhood: String, office: String,
18           place: String, postcode: String, quarter:
19             ↪ String, railway: String,
```

```

12             residential: String, retail: String, road:
13             ↪ String, shop: String, state: String,
14             suburb: String, tourism: String, town: String,
15             ↪ village: String) {
16
17     def getByName(name: String): String = {
18         try {
19             return this.getClass.getDeclaredField(name).get(this).toString
20         }
21         catch {
22             case e: Exception => return ""
23         }
24     }

```

I punti 6 e 7 sono realizzati come specificato nello snippet che segue. Di particolare rilevanza sono la `dropDuplicates("DATE", "ID_OWNER", "NAME")` e la `sort(..., desc("count"))` che assicurano quanto detto in 7.

```

1  private def best_loc_guess_all_data(dataset: Dataset[FlickrPost],
2           ↪ datasetGeo: Dataset[GeoDFItem]): DataFrame = {
3           preprocess_for_trajectory_mining(dataset).repartition(200)
4           .map(x => (x.owner.id, x.id, x.geoData.latitude,
5           ↪ x.geoData.longitude, x.dateTaken))
6           .crossJoin(
7               datasetGeo
8                   .map(x => (x.lat, x.lon, x.address.getByName(x.category) + "
9                   ↪ | " + x.address.road))
10                  .toDF("_7", "_8", "_9")
11            )
12            .map(x =>
13                (
14                    x.getFieldIndex("_1")).asInstanceOf[String],
15                    x.getFieldIndex("_2")).asInstanceOf[String],
16                    x.getFieldIndex("_7").asInstanceOf[Double],
17                    x.getFieldIndex("_8").asInstanceOf[Double],
18                    x.getFieldIndex("_9").asInstanceOf[String],
19                    dist(
20                        GeoItem(x.getFieldIndex("_3")).asInstanceOf[Double],
21                        ↪ x.getFieldIndex("_4")).asInstanceOf[Double]),
22                        GeoItem(x.getFieldIndex("_7")).asInstanceOf[Double],
23                        ↪ x.getFieldIndex("_8")).asInstanceOf[Double])
24                ),
25                x.getFieldIndex("_5")).asInstanceOf[Instant]))
26            .toDF("ID_OWNER", "ID_POST", "LAT", "LONG", "NAME", "DIST",
27            ↪ "TIMESTAMP")
28            // / ID (utente) / ID (post) / LAT / LONG / NAME / CAT / TYPE /
29            ↪ TIMESTAMP /
30            .filter(x => x.getFieldIndex("DIST")).asInstanceOf[Int] < 300)
31            .sort("ID_POST", "DIST")

```

```

24     }
25     {...}
26     private def user_loc_seq(dataset: Dataset[FlickrPost], datasetGeo:
27       → Dataset[GeoDFItem]): DataFrame =
28     {
29       val res = best_loc_guess_all_data(dataset, datasetGeo).repartition(200)
30       .withColumn("DATE", to_date(col("TIMESTAMP"), "yyyy-MM-dd"))
31       val res2 = res
32       .groupBy("NAME").count()
33       val all_sequences = res.repartition(200)
34       .join(res2, "NAME")
35       .sort(asc("DATE"), asc("TIMESTAMP"), asc("ID_OWNER"), desc("count"))
36       .dropDuplicates("DATE", "ID_OWNER", "NAME")
37       .groupBy("DATE", "ID_OWNER")
38       .agg(collect_list("NAME"))
39       .toDF("DATE", "ID_OWNER", "SEQUENCE")
40       all_sequences.write.json("sequences_json")
41       all_sequences
41   }

```

Si ricorda che `datasetGeo` è il dataset ottenuto al punto 4, il quale contiene righe come la seguente

```

1   {
2     "place_id": 256046806,
3     "licence": "Data © OpenStreetMap contributors, ODbL 1.0.
4       → https://osm.org/copyright",
5     "osm_type": "way",
6     "osm_id": 799694267,
7     "boundingbox": [
8       "41.9022186",
9       "41.9022453",
10      "12.4572349",
11      "12.4572726"
12    ],
13    "lat": "41.90223195",
14    "lon": "12.457253725772201",
15    "display_name": "Obeliscus Vaticanus, Forum Sancti Petri, Civitas Vaticana
16      → - Città del Vaticano, 00120, Civitas Vaticana - Città del Vaticano",
17    "place_rank": 30,
18    "category": "man_made",
19    "type": "obelisk",
20    "importance": 0.001,
21    "address": {
22      "man_made": "Obeliscus Vaticanus",
23      "road": "Forum Sancti Petri",
24      "city": "Civitas Vaticana - Città del Vaticano",
25      "postcode": "00120",
26      "country": "Civitas Vaticana - Città del Vaticano",
27      "country_code": "va"
28    }
29  }

```

```

26      }
27  }
```

Non tutti gli oggetti prelevati da OpenStreetMap sono risultati di interesse. In particolare, tenendo anche conto degli histogrammi relativi ai campi `category` e `typology` del dataset, in figura 3.2, si è deciso di mantenere i luoghi con valore di `category` e `typology` tra le coppie in tabella 3.1.

<code>category</code>	<code>typology</code>
amenity	monastery
amenity	art_gallery
amenity	art_gallery
amenity	place_of_worship
building	pavilion
building	monastery
building	yes
building	hotel
building	convent
building	church
building	ruins
building	public
historic	monastery
historic	church
historic	building
historic	tomb
historic	fort
historic	ruins
historic	monument
historic	memorial
historic	archaeological_site
man_made	obelisk
man_made	tower

Tabella 3.1: Coppie `category typology` di interesse

Il motivo è il seguente: come è possibile osservare in figura 3.2, la gran parte dei luoghi ricadono nella categoria *highway*, la quale rappresenta autostrade/-superstrade e sono pertanto scartate. Altre ricadono nella categoria *place*, ma,

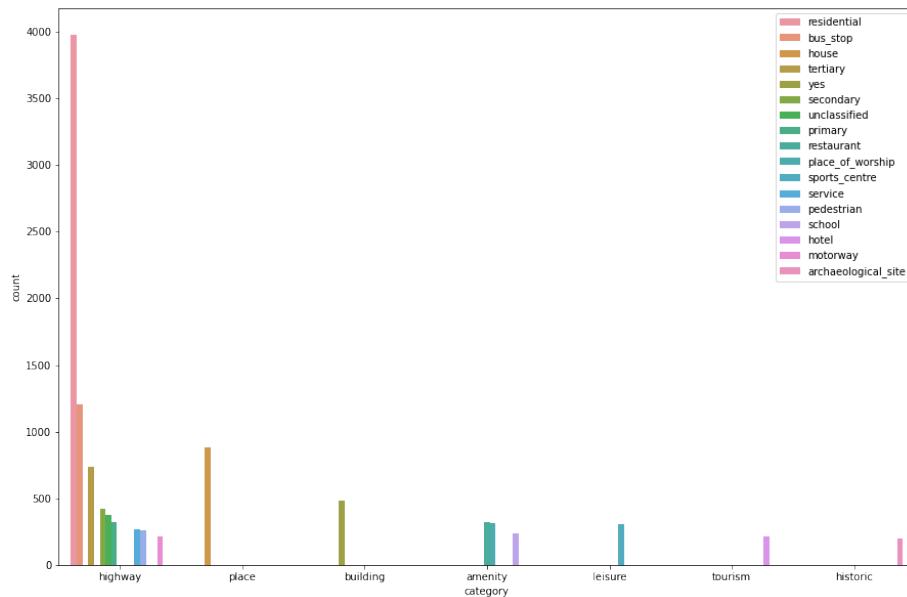


Figura 3.2: Istogrammi «category and typology »

come si può osservare dalla legenda in alto a destra, la tipologia assegnata è *house*, pertanto si tratta di abitazioni. Le categorie *building* (costruzioni), *amenity* (comodità) risultano promettenti, in quanto associate a tipologie, quali *place of worship* (lett. luogo di culto), probabilmente più visitate dagli utenti. Si includono inoltre le categorie *historic* e *man\_made* (non visibile nel grafico per via del basso numero di elementi associati). Si è scartato *tourism* in quanto associato a *hotel*.

### 3.2.3 Task 3 : Trasformazione del dataset

Il task richiede che il dataset output del task 3.2.2, il quale è in realtà un `DataFrame` con `Row` convertibili in (`Instant`, `String`, `Seq[String]`) e rappresentanti un singolo percorso effettuato da un utente in una specifica data, venga trasformato in un `Dataset[Seq[Seq[Seq[String]]]]`, dove ogni riga rappresenta un percorso svolto da un utente nell'arco di un'intera giornata.

Il `Dataset` così ottenuto (convertito in `DataFrame` con colonna `embedding`, come da specifiche di PrefixSpan) viene poi usato per ottenere i top pattern di movimento degli utenti, come riportato nello snippet seguente

```

1  def frequent_seq_pat(dataframe: DataFrame): DataFrame = {
2      val result = new PrefixSpan()
3          .setMinSupport(0.02)
4          .setMaxPatternLength(10)
5          .setMaxLocalProjDBSize(32000000)

```

```

6      .findFrequentSequentialPatterns(dataframe)
7
8      val tot = dataframe.count()
9      val freq_supp = result.map(x =>
10        (x(0).asInstanceOf[Seq[Seq[String]]], x(1).asInstanceOf[Long],
11         → x(1).asInstanceOf[Long] / (1.0 * tot)))
11      .toDF("sequence", "count", "supp")
12      → freq_supp.write.json("freq_patterns")
12      freq_supp
13    }
14  {...}
15  def topPatterns(howMany: Int = 5, filter: String = null, minLength: Int =
16    → 1): DataFrame = {
16    val freq_patterns = spark.read.json("freq_patterns").repartition(20)
17    {...}
18    .flatMap(x => x._2)
19    .map(x => (x._1.length, x._1, x._2, x._3))
20    .toDF("PATTERN_LENGTH", "PATTERN", "SUPPORT", "COUNT")
21    .sort(asc("PATTERN_LENGTH"), desc("SUPPORT"), desc("COUNT"))
22  }

```

### 3.2.4 Task 4 : Analisi e confronto dei risultati ottenuti

Si mostrano ora alcuni risultati di interesse ottenuti tramite Trajectory Mining.

Alcuni dei top pattern, selezionati anche in base ai luoghi presenti nei risultati delle analisi in [Belcastro et al. 2019], sono presenti in tabella 3.2.

Pattern	Supporto
Piazza del Campidoglio → Foro di Traiano	11.60%
Colosseo → Domus Aurea	10.22%
Colosseo → Ludus Magnus	9.59%
Basilica Sancti Petri → Braccio di Carlo Magno	10.37%
Forum Romanum → Tempio del Divo Giulio	7.52%
Musei Vaticani → San Nilammone   Forum Sancti Petri → Braccio di Carlo Magno	4.60%
Colosseo → Santi Luca e Martina al Foro Romano → Tempio della Concordia	4.02%
Basilica Sancti Petri → Palazzo del Sant'Uffizio → Braccio di Carlo Magno	10.04%
Forum Romanum → Tempio del Divo Giulio → Foro di Nerva	7.31%
Casa delle Vestali → Forum Romanum → Foro di Nerva	6.51%
Colosseo → Ludus Magnus → Antiquarium del Celio → Domus Aurea	6.45%
Casa delle Vestali → Forum Romanum → Tempio del Divo Giulio → Necropoli arcaica	6.98%
Piazza Venezia → Santa Maria in Aracoeli → Insula dell'Ara Coeli → Foro di Traiano	7.03%
Obeliscus Vaticanus → Basilica Sancti Petri → Palazzo del Sant'Uffizio → Braccio di Carlo Magno	8.33%
Tipografia Vaticana → Basilica Sancti Petri → San Nilammone → Braccio di Carlo Magno	7.69%

Tabella 3.2: Alcuni pattern di interesse

Si possono notare differenze con i risultati in [Belcastro et al. 2019]: ciò è dovuto al differente meccanismo di associazione di luoghi agli utenti. In particolare, per via dei filtri applicati con la tabella 3.1 si sono persi alcuni luoghi presenti invece in [Belcastro et al. 2019], quali ad esempio il Pantheon e la Fontana di Trevi.

## Capitolo 4

# User Clustering

Si è svolta un’analisi di clustering - basata su contenuto - sugli utenti della piattaforma, mirata a individuare categorie di utenti e suggerire, sulla base del clustering effettuato e dell’opportuna rappresentazione vettoriale dell’utente, *tag* e contenuti agli utilizzatori di Flickr.

Un’analisi di questo tipo può essere svolta in diversi modi. Tuttavia, sono sempre necessari i seguenti task:

- **Task 1:** *mapping* degli utenti in uno spazio vettoriale, in modo da applicare algoritmi di Machine Learning;
- **Task 2 :** scelta del meccanismo di clustering e delle metriche opportune;
- **Task 3 :** individuazione dei parametri che rendono migliore il modello;
- **Task 4 :** l’interpretazione dei risultati.

Verranno ora descritte le modalità con le quali sono stati affrontati i task sopra elencati.

### 4.1 Task 1 : User Embedding

Il *mapping* di oggetti di qualsiasi natura in uno spazio vettoriale viene detto *embedding*.

Un *embedding* è una funzione da un dominio  $D$  a un dominio in  $\mathbb{R}^n$ , non necessariamente invertibile. L’obiettivo di un embedding, in particolare nell’ambito del *Natural Language Processing*, è mappare oggetti semanticamente correlati in zone adiacenti nello spazio di embedding. Similmente, oggetti semanticamente distanti dovrebbero risultare tali nello spazio di embedding.

Nel caso in esame, il dominio  $D$  è costituito da utenti di Flickr, i quali interagiscono, e quindi espongono come proprie caratteristiche, solo i *post* caricati

sul sito. Si può pensare pertanto di caratterizzare ogni utente con l'insieme dei post di sua creazione. A loro volta, i post sono caratterizzati da:

- un *titolo*;
- una *descrizione*, in genere non particolarmente lunga;
- una *foto*;
- altre caratteristiche.

Sebbene sia possibile anche in Spark estrarre *feature* significative da foto, come sarà mostrato in seguito, si è deciso di mantere come caratteristiche dei post, a fini dell'embedding degli utenti, i soli *titolo* e *descrizione*. In particolare, sono state filtrate tutte le istanze di post nei quali mancava uno dei due attributi (o entrambi).

Il problema diventa dunque trovare un embedding opportuno per i post di un utente.

Esistono diversi approcci validi rispetto al problema:

- si possono sfruttare modelli preallenati, come Doc2Vec [Le e Mikolov 2014], in grado di fornire embedding di *documenti*, o sequenze di parole;
- si può allenare un modello *ad hoc*, come LDA [Blei, Ng e Jordan 2003], per inferire embedding non solo dei singoli post, ma anche dei *topic* che li caratterizzano.

Alcuni modelli, tuttavia, si prestano meglio di altri al contenuto dei post degli utenti. In particolare, i post degli utenti Flickr sono caratterizzati da titoli e descrizioni particolarmente brevi, il che pregiudica l'utilizzo di modelli come LDA o Doc2Vec nelle loro versioni usuali.

Inoltre, una problematica rilevante è la seguente: i post nel dataset sono **eterogenei** per quanto riguarda la lingua utilizzata. Un modo per aggirare il problema potrebbe essere quello di introdurre un modello di traduzione nella *pipeline* per la generazione degli embedding. Tuttavia, ciò introduce non solo *overhead*, ma anche notevole *rumore*, dovuto al fatto che i task di traduzione non sono affatto semplici.

Si è deciso pertanto, dopo alcuni tentativi, di optare per un modello *language agnostic*: il *Multilingual Universal Sentence Encoder* [Yang et al. 2019].

Un modello preallenato è disponibile su Spark NLP<sup>1</sup>.

Una volta ottenuti gli embedding post per post di tutti gli utenti, si definisce l'embedding di un singolo utente nel modo seguente:

---

<sup>1</sup> *Universal Sentence Encoder Multilingual Large (tfhub\_use\_multi\_lg)- Spark NLP Model* (2022). URL: [https://nlp.johnsnowlabs.com/2021/05/06/tfhub\\_use\\_multi\\_lg\\_xx.html](https://nlp.johnsnowlabs.com/2021/05/06/tfhub_use_multi_lg_xx.html) (visitato il 06/02/2022).

$$\text{embedding}(\text{user}) = \frac{\sum_{x \in \text{posts}(\text{user})} \text{embedding}(x)}{|\text{posts}(\text{user})|}$$

dove  $\text{posts}(\text{user})$  rappresenta l'insieme dei post generati dall'utente  $\text{user}$ .

Si può notare come lo spazio di embedding tra utenti e post sia pertanto condiviso: ciò permette ad esempio di suggerire ad un utente la visualizzazione di un post sulla base della loro distanza nello spazio latente (o degli embedding).

La normalizzazione ci permette di utilizzare come concetto di distanza la distanza euclidea, invece di distanze come distanza basata su coseno.

Si riporta uno snippet contenente il codice usato per l'embedding degli utenti

```

1  def userEmbeddings(dataset: Dataset[FlickrPost], normalize: Boolean = true):
2      → DataFrame = {
3          val document = new DocumentAssembler()
4              .setInputCol("text")
5              .setOutputCol("document")
6          val embedder = UniversalSentenceEncoder.pretrained("tfhub_use_multi",
7              ← "xx")
8              .setInputCols("document")
9              .setOutputCol("sentence_embeddings")
10         val embeddingsFinisher = new EmbeddingsFinisher()
11             .setInputCols("sentence_embeddings")
12             .setOutputCols("finished_embeddings")
13             .setOutputAsVector(true)
14         val pipeline = new Pipeline().setStages(Array(
15             document,
16             embedder,
17             embeddingsFinisher)
18         )
19         val documentDF = dataset.filter(x => (x.title != null && x.description
20             ← != null)).repartition(200)
21             .map(x => (x.owner.id, get_text(x.title, x.description)))
22             .toDF("user", "text")
23         val embeddings = pipeline.fit(documentDF).transform(documentDF)
24             .selectExpr("user", "text", "explode(finished_embeddings) as
25             ← embedding")
26         val user_embeddings =
27             embeddings.groupByKey(x => x(0).toString).mapValues(x =>
28                 ← (x(2).asInstanceOf[DenseVector].toArray.toSeq, 1))
29             .reduceGroups((a, b) => (a._1.zip(b._1).map(c => c._1 + c._2), a._2
30                 ← + b._2))
31         user_embeddings.write.json("post_plus_embedding")
32         if (!normalize)
33             user_embeddings.map(x => (x._1, x._2._1))
34                 .toDF("user", "embedding")
35         else

```

```

30     user_embeddings.map(x => (x._1, x._2._1.map(z => z / (1.0 *
31         ← x._2._2))))
32     .toDF("user", "embedding")
33 }
```

## 4.2 Task 2 e 3: Clustering e selezione parametri

Esistono diverse tecniche di clustering applicabili al task di interesse. Si è deciso di optare per KMeans, in quanto Spark MLlib<sup>2</sup> mette a disposizione una implementazione parallela dell'algoritmo.

Di seguito lo snippet contenente il codice relativo a KMeans:

```

1
2 def load_user_embeddings(normalize: Boolean): Dataset[UserEmbedding] = {
3     val user_embeddings = spark.read.json("post_plus_embedding")
4     .as[(Tuple2[Seq[Double], Long], String)]
5     if (!normalize)
6         user_embeddings.map(x => (x._2, x._1._1))
7         .toDF("user", "embedding").as[UserEmbedding]
8     else
9         user_embeddings.map(x => (x._2, x._1._1.map(z => z / (1.0 *
10            ← x._1._2))))
11         .toDF("user", "embedding").as[UserEmbedding]
12 }
13
14 def userClustering(k: Int, embeddings: DataFrame): KMeansModel = {
15     val kmeans = new KMeans()
16     .setK(k)
17     .setSeed(5L)
18     .setMaxIter(100)
19     .setInitSteps(2)
20     .setInitMode("k-means||")
21     .setFeaturesCol("embedding")
22     .setPredictionCol("prediction")
23     val kmeansModel = kmeans.fit(embeddings)
24     kmeansModel
25 }
```

Per quanto riguarda la scelta dell'iperparametro  $k$  di KMeans, si è svolta un'analisi dell'andamento dell'*inerzia* al variare del parametro  $k$ , usando l'*elbow method* (lett. metodo del gomito).

In figura 4.1 è presente l'andamento. Si è scelto pertanto il parametro  $k = 6$ .

---

<sup>2</sup>Clustering - Spark 3.2.1 Documentation (2022). URL: <https://spark.apache.org/docs/latest/ml-clustering.html#k-means> (visitato il 06/02/2022).

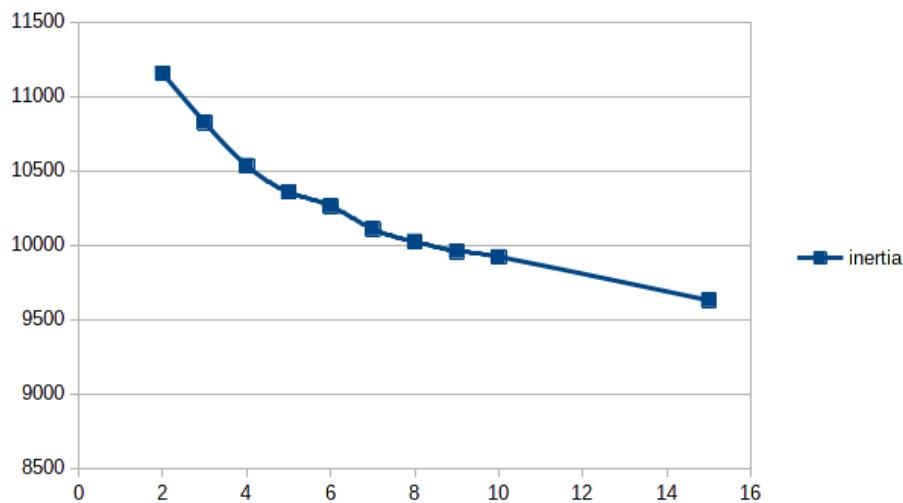


Figura 4.1: Andamento dell'inerzia

### 4.3 Task 4 : Interpretazione dei risultati

Una volta alleanto il modello KMeans con il parametro  $k = 6$ , è stato possibile individuare, per ogni cluster, alcuni rappresentanti. Per rappresentanti di cluster si intendono utenti i cui embedding risultano vicini al centro di massa (o centroide) del cluster di appartenenza.

Per il cluster 0, sono stati individuati i seguenti rappresentanti:

Utente	Parole più usate
98274023@N00	vatican, st, rome, museum, i, pantheon, hotel, peter, paul, new
32076237@N00	rome, the, san, basilica, pietro, saint, peter, vatican, piazza, colosseum
44192643@N02	peter, st, vatican, square, santa, rome, maria, dome, inside, colosseum
83031170@N00	the, rome, st, peter, inside, vatican, forum, basilica, fountain, coliseum
22094057@N05	the, peter, st, vatican, coliseum, basilica, fountain, trevi, museum, forum

Tabella 4.1: Rappresentanti cluster 0

Gli utenti appaiono alquanto eterogenei, accomunati principalmente dalla descrizione e visita di luoghi quali la Basilica di San Pietro e il Colosseo.

Per il cluster 1:

<b>Utente</b>	<b>Parole più usate</b>
55391611@N00	rome, the, basilica, vatican, city, peter, saint, santa, maria, church
22158962@N07	roma, san, org, santa, wikipedia, http, href, wiki, www, piazza
28353725@N00	rome, i, the, piazza, street, quot, church, it, one, nuns
77547214@N00	the, rome, fountain, roman, forum, temple, colosseum, quot, st, peter
25718393@N04	quot, the, rome, roma, wikipedia, href, com, see, http, flickr

Tabella 4.2: Rappresentanti cluster 1

Gli utenti del cluster 1, come è possibile osservare consultando i post dei rappresentanti, appaiono particolarmente descrittivi, citando articoli e wikipedia quando possibile.

Per il cluster 2:

<b>Utente</b>	<b>Parole più usate</b>
34857532@N00	en, wiki, org, wikipedia, rome, http, href, i, villa
24793644@N08	crunch, i, rome, dsc, jpg, foto, storico, 5, b,
63327992@N07	i, a, the, rome, old, nice, our, fountain, water, maxentius
19446102@N00	rome, quot, href, http, i, rome, rel,nofollow, via, piazza
33399095@N00	roma, www, com, href, http, flickr, quot, photos, e, mm

Tabella 4.3: Rappresentanti cluster 2

Gli utenti del cluster 2 sono meno descrittivi degli utenti del cluster 1. Inoltre, includono link a altri siti, e sembrano utilizzare un linguaggio a volte filosofico/poetico.

Per il cluster 3

<b>Utente</b>	<b>Parole più usate</b>
8099187@N06	com, omogirando, href, rel,nofollow, www, http, jimdo, b, facebook
69912818@N00	href, http, rel,nofollow, www, com, b, rome, sound36
11432907@N00	href, http, com, rel,nofollow, large, amp, bighugelabs, onblack, php
11102419@N00	href, http, org, wikipedia, wiki, en, rome, com, flickr, the
21336230@N08	href, http, rel,nofollow, com, rome, www, amp, view, large

Tabella 4.4: Rappresentanti cluster 3

Gli utenti del cluster 3 sono caratterizzati praticamente da URL. Potrebbero essere considerati spammer nella piattaforma, anche se ciò non è propriamente corretto, come sarà mostrato a breve.

Per il cluster 4:

<b>Utente</b>	<b>Parole più usate</b>
29223649@N04	de, roma, piazza, san, via, en, n, 2, y, palazzo
96291012@N00	img, com, www, bertolinidennis, myspace, 20110602, http, href, 06, edited
47211255@N05	img, a, s, day, piazza, navona, rome, lina, ivo, not
58826214@N00	column, trajan, roman, rome, aurelius, marcus, it, built, the, spiral
25538307@N00	dsc, img, 8, marzo, eucalipti, minirugby, ios, u16, s, photos

Tabella 4.5: Rappresentanti cluster 4

Si tratta di utenti pigri da un punto di vista testuale, con descrizioni brevissime e titoli inefficaci. Anche di singole parole o di semplici spazi. Gli argomenti sembrano vari, anche se si può notare la presenza della parola *img*, in quanto in genere contenuta nei nomi di default di foto.

Infine, per il cluster 5:

<b>Utente</b>	<b>Parole più usate</b>
63558118@N07	voigtlander, 5, f, rome, 20mm, la, d200, valeria, rome, steps
136373368@N02	rome, rome, gh4, panasonic, picture, villa, autumn, borgheze, shooted
13958243@N08	ilford, rome, nikon, epson, v750, kodak, nikkor, quot, tmax, d76
33920763@N08	de, ce, societ, 1, nikon, n, embe, famo, vino, statue
27818145@N00	http, href, kodak, com, 2, f, planar, 80mm, hasselblad, 501cm

Tabella 4.6: Rappresentanti cluster 5

Si nota immediatamente la predominanza di termini di natura tecnica riguardanti fotocamere e obiettivi. Sembra pertanto un cluster di utenti appassionati alla fotografia, più che alla città di Roma.

I cluster di utenti si differenziano inoltre anche sulla base di alcune statistiche, riportate in figura 4.2.

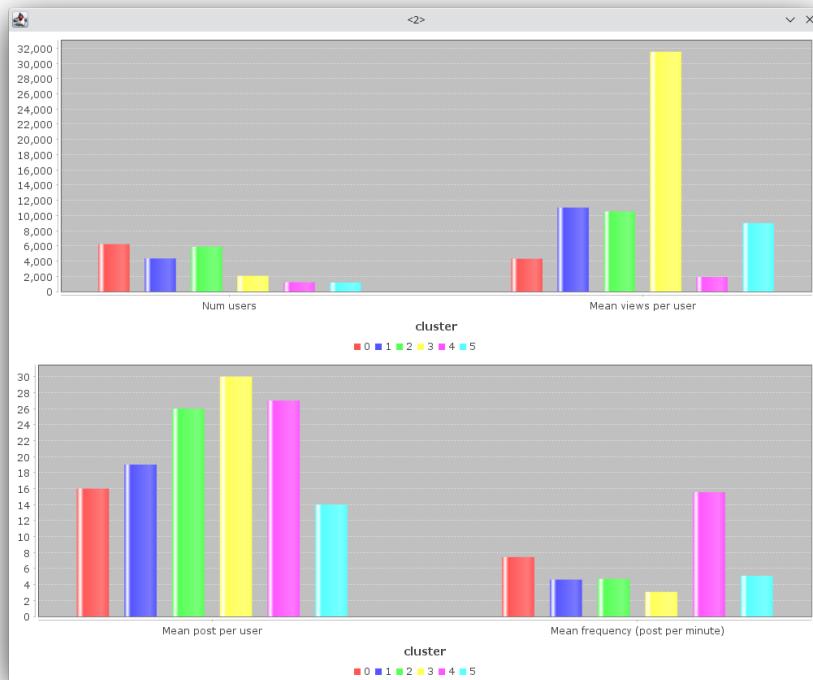


Figura 4.2: Alcune statistiche relative ai cluster individuati

Si può notare quanto sia elevato, in proporzione agli altri cluster, il numero di utenti che ricadono nei cluster 0, 1 e 2. Gli utenti del cluster 3 sembrano avere un ottimo numero di visualizzazioni, ma anche un numero elevato di post medio, con bassa frequenza per post<sup>3</sup> (che indica una maggiore permanenza su Flickr, pertanto si tratta di utenti longevi).

Diversamente, gli utenti del cluster 4 risultano essere spammer, o quasi, in quanto la loro media di post è paragonabile o superiore a quella degli altri cluster, tuttavia posseggono una frequenza di post al minuto molto maggiore.

Gli appassionati di fotografia, ossia gli utenti del cluster 5, più vicini in un certo senso allo spirito di Flickr, risultano avere un buon numero di visualizzazioni rispetto alla frequenza di post, il che indica che la loro capacità, anche di artisti della fotografia, è in qualche modo premiata dagli utenti della piattaforma.

<sup>3</sup>La frequenza per post (al minuto) è calcolata come il numero di post totali di quell'utente, diviso per il numero di minuti trascorsi tra il meno recente e il più recente post dell'utente

## 4.4 Ulteriori elementi di categorizzazione

La libreria SynapseML di Microsoft<sup>4</sup> permette di utilizzare, in maniera distribuita, modelli di Machine Learning *pretrained* per diversi task, anche riguardanti *image classification* e *image segmentation*. Ciò è possibile grazie allo standard ONNX (Open Neural Network Exchange)<sup>5</sup>, il quale permette una migrazione (quasi) indolare di modelli di machine learning su piattaforme e sistemi differenti, come Python e Scala.

Si è pensato, come *proof of concept*, di utilizzare la libreria SynapseML in congiunzione con un modello ONNX per l'*image classification*: in particolare, il modello GoogleNet<sup>6</sup>. Si è tentato anche di utilizzare un modello più potente e adatto agli scopi, InceptionV2<sup>7</sup>, tuttavia alcune limitazioni di Scala (relative ad un tipo di dati utilizzato dalla sottostante libreria *tensorflow*) non ne ha permesso l'utilizzo.

L'idea è la seguente: attraverso classificazione di immagini, fornire feature (anche basate su clustering di immagini) aggiuntive che descrivano il post di un utente, oltre ai semplici titolo e descrizione.

Il modello GoogleNet è in grado di individuare le 1000 classi di ImageNet<sup>8</sup>, anche se le performance degradano al crescere del numero di oggetti nell'immagine in input. Tuttavia, non è stato possibile, per motivi di tempo e di spazio, scaricare per ogni post di ogni utente la relativa foto, per cui l'idea non è stata implementata in toto.

In figura 4.3 si mostra un esempio dell'output di GoogleNet su una foto prelevata da un post del dataset.

---

<sup>4</sup> *Synapse Machine Learning* (6 feb. 2022). original-date: 2017-06-05T08:23:44Z. URL: <https://github.com/microsoft/SynapseML> (visitato il 06/02/2022).

<sup>5</sup> *ONNX Model Zoo* (6 feb. 2022). original-date: 2017-10-06T00:03:03Z. URL: <https://github.com/onnx/models> (visitato il 06/02/2022).

<sup>6</sup> *ONNX Model Zoo* (6 feb. 2022). original-date: 2017-10-06T00:03:03Z. URL: <https://github.com/onnx/models> (visitato il 06/02/2022).

<sup>7</sup> *ONNX Model Zoo* (6 feb. 2022). original-date: 2017-10-06T00:03:03Z. URL: <https://github.com/onnx/models> (visitato il 06/02/2022).

<sup>8</sup> *IMAGENET 1000 Class List - WekaDeepLearning4j* (2022). URL: <https://deeplearning.cms.waikato.ac.nz/user-guide/class-maps/IMAGENET/> (visitato il 06/02/2022).

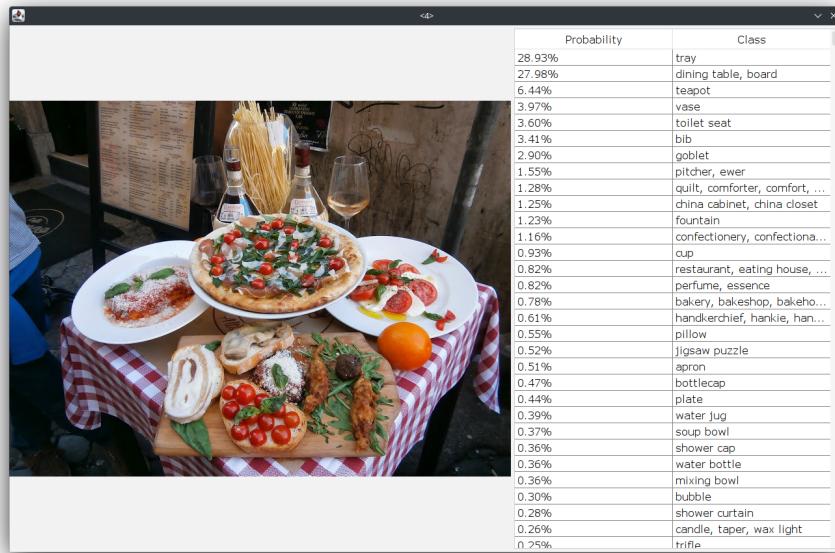


Figura 4.3: Esempio GoogleNet

Di seguito è presente uno snippet che mostra il caricamento da file del modello ONNX e l'esecuzione del modello stesso su una foto in input:

```

1
2  def imageNet(filename: String): Array[Seq[(Float, String)]] = {
3    val onnx = new ONNXModel().setModelLocation("googlenet-12.onnx")
4    val inputImg = spark.read.format("image")
5      .load("images/" + filename)
6    val image_df = (new ImageTransformer()
7      .setInputCol("image")
8      .setOutputCol("transformed_images")
9      .resize(224, true)
10     .centerCrop(224, 224)
11     .setNormalizeMean(Array(-123.68, -116.779, -103.939))
12     .setToTensor(true)
13     .transform(inputImg))
14    val m = onnx.setDeviceType("CPU")
15      .setFeedDict("data_0", "transformed_images")
16      .setFetchDict("probability", "prob_1")
17      .setMiniBatchSize(1)
18    val probs = m.transform(image_df).coalesce(1).select("probability")
19      .map(x => x(0).asInstanceOf[Seq[Float]]).collect()
20    val imageNetClasses = spark.read.option("header",
21      "true").csv("imagenet.csv")
22      .select("class")
23      .map(x => x(0).toString)
24      .collect()

```

```

24     probs.map(x => x.zip(imageNetClasses).sortBy(_._1).reverse)
25   }
26

```

È stato inoltre possibile integrare un modello di Spark NLP<sup>9</sup> in grado di riconoscere e *annotare* i post degli utenti in base alla lingua. Anche questa feature poteva essere utilizzata per un clustering più appropriato. In figura 4.4 si mostrano alcuni esempi su tre post di lingue diverse.

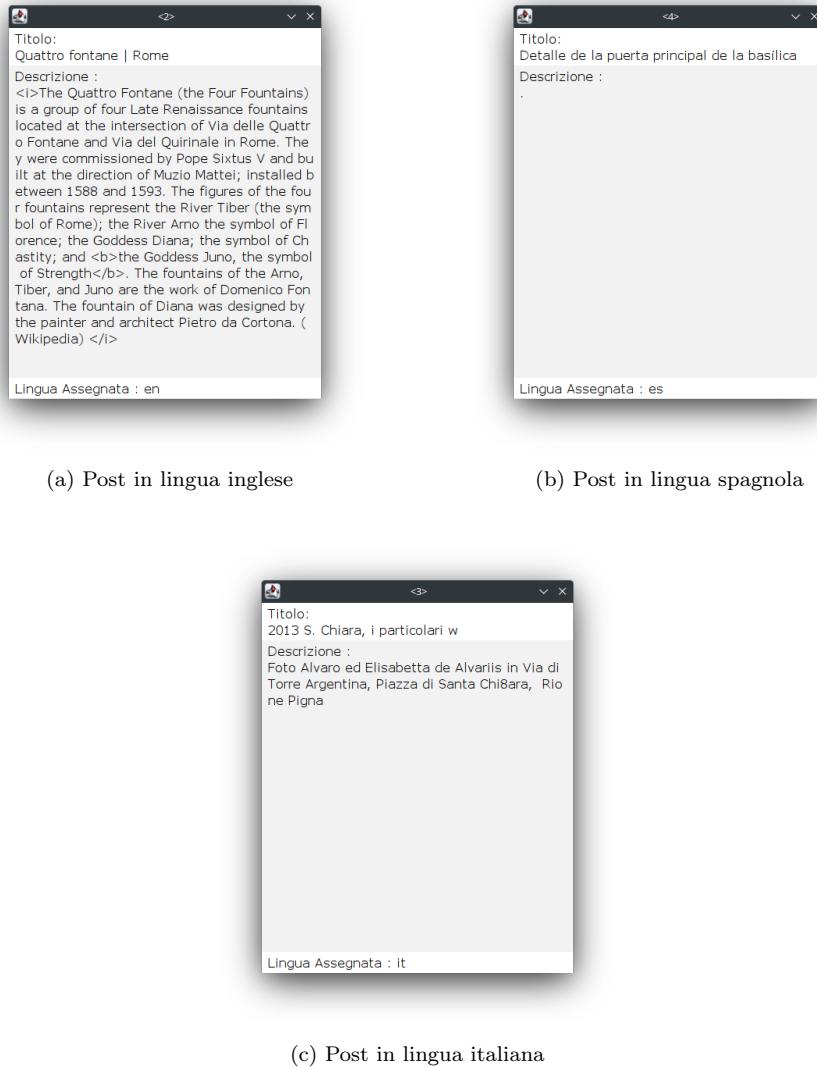


Figura 4.4: Esempio language detection

---

<sup>9</sup> *Language Detection & Identification Pipeline - 21 Languages- Spark NLP Model* (2022). URL: [https://nlp.johnsnowlabs.com/2020/12/05/detect\\_language\\_21\\_xx.html](https://nlp.johnsnowlabs.com/2020/12/05/detect_language_21_xx.html) (visitato il 06/02/2022).

## Capitolo 5

# Analisi delle prestazioni su dati sintetici

È stata svolta un'analisi delle prestazioni rispetto ad alcuni task di interesse sui dataset sintetici, in modo da valutare la scalabilità del sistema all'aumentare del numero di tuple nel dataset.

I task analizzati sono i seguenti:

- task 1: andamento dei post di un utente per anno;
- task 2: utenti più influenti sulla base di visualizzazioni e post pubblicati (score);
- task 3: andamento dei tag negli anni;
- task 4: distribuzione dei tag nel dataset;

I dataset sintetici utilizzati hanno dimensioni crescenti e dipendenti dalla dimensione del dataset originale. In particolare, sono stati utilizzati 3 dataset sintetici:

- **flickr2x.json**, un dataset di dimensioni pari a 2 volte (2x) il dataset originale;
- **flickr4x.json**, un dataset di dimensioni pari a 4 volte (4x) il dataset originale;
- **flickr8x.json**, un dataset di dimensioni pari a 8 volte (8x) il dataset originale.

Il grafico risultante dall'analisi è presente in figura 5.1. Come si può notare, l'andamento dei tempi totali richiesti dai task segue un andamento pressochè lineare al crescere della dimensione del dataset.

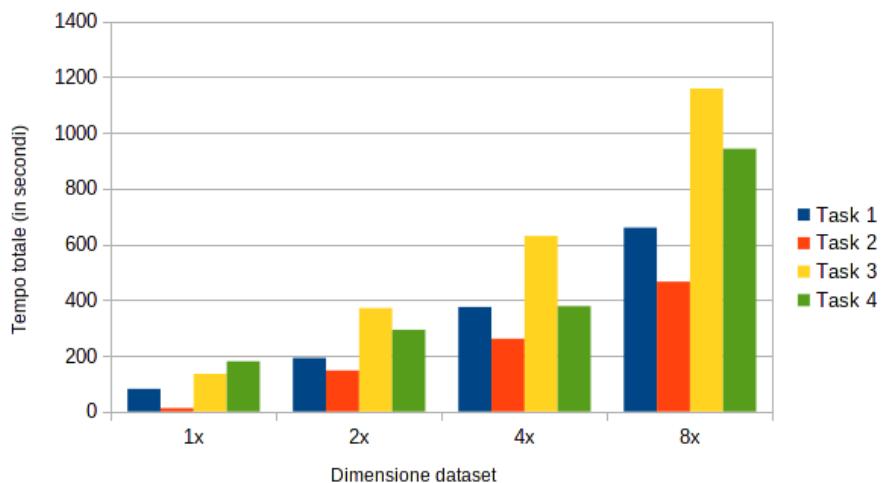


Figura 5.1: Andamento dei tempi al crescere della cardinalità del dataset

# Bibliografia

- Belcastro, Loris et al. (dic. 2019). “ParSoDA: high-level parallel programming for social data mining”. In: *Social Network Analysis and Mining* 9.1, p. 4. ISSN: 1869-5450, 1869-5469. DOI: 10 . 1007 / s13278 - 018 - 0547 - 5. URL: <http://link.springer.com/10.1007/s13278-018-0547-5> (visitato il 03/02/2022).
- Blei, David M., Andrew Y. Ng e Michael I. Jordan (2003). “Latent Dirichlet Allocation”. In: *Journal of Machine Learning Research* 3 (Jan), pp. 993–1022. ISSN: ISSN 1533-7928. URL: <https://www.jmlr.org/papers/v3/blei03a> (visitato il 07/02/2022).
- Clustering - Spark 3.2.1 Documentation* (2022). URL: <https://spark.apache.org/docs/latest/ml-clustering.html#k-means> (visitato il 06/02/2022).
- Frequent Pattern Mining - Spark 3.2.1 Documentation* (2022). URL: <https://spark.apache.org/docs/latest/ml-frequent-pattern-mining.html#prefixspan> (visitato il 04/02/2022).
- IMAGENET 1000 Class List - WekaDeepLearning4j* (2022). URL: <https://deeplearning.cms.waikato.ac.nz/user-guide/class-maps/IMAGENET/> (visitato il 06/02/2022).
- Language Detection & Identification Pipeline - 21 Languages- Spark NLP Model* (2022). URL: [https://nlp.johnsnowlabs.com/2020/12/05/detect\\_language\\_21\\_xx.html](https://nlp.johnsnowlabs.com/2020/12/05/detect_language_21_xx.html) (visitato il 06/02/2022).
- Le, Quoc V. e Tomas Mikolov (22 mag. 2014). “Distributed Representations of Sentences and Documents”. In: *arXiv:1405.4053 [cs]*. arXiv: 1405 . 4053. URL: <http://arxiv.org/abs/1405.4053> (visitato il 07/02/2022).
- Measuring accuracy of latitude and longitude?* (2022). Geographic Information Systems Stack Exchange. URL: <https://gis.stackexchange.com/questions/8650/measuring-accuracy-of-latitude-and-longitude> (visitato il 04/02/2022).
- ONNX Model Zoo* (6 feb. 2022). original-date: 2017-10-06T00:03:03Z. URL: <https://github.com/onnx/models> (visitato il 06/02/2022).

- Overview - Nominatim Documentation* (2022). URL: <https://nominatim.org/release-docs/develop/api/Overview/> (visitato il 04/02/2022).
- Pei, Jian et al. (nov. 2004). “Mining sequential patterns by pattern-growth: the PrefixSpan approach”. In: *IEEE Transactions on Knowledge and Data Engineering* 16.11. Conference Name: IEEE Transactions on Knowledge and Data Engineering, pp. 1424–1440. ISSN: 1558-2191. doi: 10.1109/TKDE.2004.77.
- Servizi Flickr (2022). URL: <https://www.flickr.com/services/api/> (visitato il 07/02/2022).
- Synapse Machine Learning* (6 feb. 2022). original-date: 2017-06-05T08:23:44Z. URL: <https://github.com/microsoft/SynapseML> (visitato il 06/02/2022).
- Universal Sentence Encoder Multilingual Large (tfhub\_use\_multi\_lg)- Spark NLP Model* (2022). URL: [https://nlp.johnsnowlabs.com/2021/05/06/tfhub\\_use\\_multi\\_lg\\_xx.html](https://nlp.johnsnowlabs.com/2021/05/06/tfhub_use_multi_lg_xx.html) (visitato il 06/02/2022).
- Yang, Yinfel et al. (9 lug. 2019). “Multilingual Universal Sentence Encoder for Semantic Retrieval”. In: *arXiv:1907.04307 [cs]*. arXiv: 1907.04307. URL: <http://arxiv.org/abs/1907.04307> (visitato il 06/02/2022).