Università degli studi di Trento

Dipartimento di Ingegneria e Scienza dell'Informazione

Corso di Laurea triennale in INFORMATICA

Tesi Finale

Clustrify: clustering non supervisionato di tweet

Relatore
Prof. Alberto Montresor

Laureando
Mattia Larentis

Anno accademico 2013-2014

Dedicata a Michela

Indice

Prefazione **4**

1 Clusters e clustering **5**
1.1 Clustering partizionale 6
1.2 Clustering gerarchico 7
1.3 Clustering spectral 8
1.4 Altri modelli di clutering 8

2 Clusterify **10**
2.1 Background 10
2.1.1 Twitter API 10
2.1.1.1 REST API v1.1 11
2.1.2 dataTXT 11
2.1.2.1 dataTXT NEX 12
2.1.2.2 dataTXT REL 13
2.2 Backend 14
2.2.1 Algoritmi di clustering 15
2.2.1.1 K-means 15
2.2.1.2 Spectral 16
2.2.1.3 Affinity Propagation 17
2.2.1.4 Ward 18
2.3 Scelta dell'algoritmo di clustering 18
2.4 Workflow 19
2.4.1 Acquisizione dei testi 20
2.4.2 Annotazioni dei testi 20
2.4.3 Clustering dei testi 20
2.4.3.1 Creazione matrice di adiacenza 21
2.4.3.2 Esecuzione dell'algoritmo di clustering 21
2.5 Frontend 21
2.5.1 Studio iniziale 21
2.5.2 Studio finale 25

Conclusioni e sviluppi futuri **30**

Ringraziamenti **31**

Prefazione

> "È evidente che l'uomo sia un essere sociale più di ogni ape e più di ogni animale da gregge. Infatti, la natura non fa nulla, come diciamo, senza uno scopo: l'uomo è l'unico degli esseri viventi a possedere la parola.
>
> — Aristotele, Politica

L'uomo, in quanto essere sociale, per natura è portato al bisogno di comunicare. Dall'invenzione del telegrafo all'utilizzo di Internet è cambiato solamente il mezzo di divulgazione, ma non l'atto di voler condividere pensieri con il resto dell'umanità.

Questi pensieri vengono talvolta espressi sotto forma di testi, di dimensione variabile, che un individuo scrive nella speranza che da altri vengano letti. La grande mole di messaggi può essere elaborata per ricavarne informazioni utili, potenzialmente per la risoluzione di alcuni problemi.

Clusterify è un'applicazione web che mira alla possibilità di risolvere più problemi contemporaneamente: dividere un ammasso di in insiemi distinti, dove ognuno è caratterizzato da un forte legame che le proprie componenti hanno tra di loro, e da un legame debole nei confronti delle altre. Questo processo può aprire le porte a molteplici risultati, come la possibilità di filtrare testi per una macro-area interessata, piuttosto che caratterizzare persone interessate ad un'attività per capire come meglio investire nel prossimo semestre.

Il mio progetto ha l'intento di mostrare la potenzialità del clustering non supervisionato per la suddivisione di testi. Nella fattispecie si ha intenzione di applicare degli algoritmi di clustering per definire questi sottogruppi ed, una volta composti, utilizzarli come strumento per filtrare gli stessi messaggi. Supponendo che l'utente medio di Twitter non parli solamente dell'argomento per cui è stato seguito, è possibile che questo scriva di materie al nostro utente interessanti o, al contrario, ininfluenti.

Clusterify, così facendo, permette di leggere solamente per lui interessanti.

Capitolo 1

Clusters e clustering

Con il nome di *clustering* si definisce la tecnica non supervisionata utile al raggruppamento di oggetti in insiemi secondo determinate regole. Gli elementi appartenenti allo stesso *cluster* sono più simili tra loro rispetto a quelli contenuti negli altri gruppi. La *cluster analysis* è diventata una componente molto importante nell'analisi dei dati, sia in campo scientifico che industriale. Quando un insieme di dati deve essere organizzato, questa tecnica assegna ogni singolo dato ad un gruppo utilizzando strutture presenti nei *row data*.

L'apprendimento non supervisionato è una tecnica che consiste nel fornire al sistema una serie di dati in input che, successivamente, riclassificherà ed organizzerà sulla base di caratteristiche che questi oggetti hanno in comune senza avere bisogno di ulteriori informazioni.

Queste regole tendono a minimizzare la *distanza* interna a ciascun gruppo ed a massimizzare quella tra i gruppi stessi.

La distanza (o *metrica*) è una qualsiasi funzione $d : X \times X \to \mathbb{R}$ che soddisfa [1]:

$$\begin{aligned} &d(x, y) \geq 0 \\ &d(x, y) = 0 \iff x = y \\ &d(x, y) = d(y, x) \end{aligned}$$

La scelta della metrica influenza drasticamente la forma dei cluster poiché i rapporti tra le varie distanze possono cambiare totalmente. Tra le metriche più comuni sono presenti la *distanza euclidea* e la *distanza di Manhattan*.

Non esiste un algoritmo in grado di raggruppare qualsiasi dato in modo più corretto rispetto ad un altro, infatti esistono tecniche specifiche per ogni tipologia di dato, da quello statistico a quello sociale.

Ci si può ispirare a molteplici nozioni per il raggruppamento: dalla creazione di gruppi caratterizzati da una piccola distanza tra i singoli membri, all'utilizzo di particolari distribuzioni statistiche.

Molti degli algoritmi usati per fare clustering godono di alcuni tratti in comune che ci portano a definire i *modelli*.

I modelli di clustering tipici sono:

- Partizionale
- Gerarchico
- Spectral
- Altri modelli

1.1 Clustering partizionale

Gli algoritmi di clustering di questa famiglia creano una partizione delle osservazioni minimizzando la funzione di costo:

$$\sum_{i=1}^{k} E(C_i)$$

dove k è il numero dei cluster richiesti in output, C_i è l'i-esimo cluster e $E : C \to R^+$ è la funzione di costo associata al singolo cluster.

Questa funzione viene spesso tradotta in [2]:

$$E(C_i) = \sum_{j=1}^{|C_i|} dist(x_j, center(i))$$

dove $|C_i|$ è il numero di oggetti presenti nel i-esimo cluster, $x_j \in C_i$ e $dist(x_j, center(i))$ è una funzione che calcola la distanza tra il punto x_j ed il centro del cluster i-esimo.

Questa tipologia di algoritmi solitamente richiede di specificare il numero di cluster distinti che si vogliono raggiungere a processo terminato e mira ad identificare i gruppi naturali presenti nel dataset, generando una partizione composta da cluster disgiunti la cui unione forma il dataset originale.

Questo significa che ogni cluster ha almeno un elemento e che un elemento appartiene ad un solo cluster.

Per segmentare l'insieme in sottogruppi si utilizza il concetto di centri: inizialmente sono posizionati in modo casuale, o secondo un qualsivoglia algoritmo, e iterativamente mossi fino a che questi non raggiungano uno stato di fermo. Quando ciò accade si è in grado di definire a quale centro viene associato il singolo punto e, di conseguenza, la struttura dei cluster calcolati.

Gli algoritmi più famosi appartenenti questa categoria sono:

- K-means

- K-medoids
- Affinity Propagation

1.2 Clustering gerarchico

Gli algoritmi di clustering gerarchico creano una rappresentazione gerarchia ad albero dei cluster. Le strategie sono tipicamente di due tipi:

- Metodo agglomerativo *(bottom-up)*
  Si inizia col creare ed associare un cluster ad ogni oggetto, e si procede poi con l'unione di questi, basando la selezione degli insiemi da unire su una *funzione di similarità.*
- Metodo divisivo *(top-down)*
  Partendo da un unico cluster contenente tutti li oggetti, lo si divide basando la scelta della selezione dell'insieme da dividere su una *funzione di similarità.*

Questa tipologia di algoritmi necessita anche di alcuni criteri di collegamento che specificano la dissimilarità di due insiemi utilizzando la distanza valutabile tra gli stessi. Questi criteri possono essere:

- Complete linkage: distanza massima tra elementi appartenenti a due cluster
$$\max \{ d(a,b) : a \in C_1,\, b \in C_2 \}$$
- Minimum o single-linkage: distanza minima tra elementi appartenenti a cluster diversi
$$\min \{ d(a,b) : a \in C_1,\, b \in C_2 \}$$
- Average linkage: la media delle distanze dei singoli elementi
$$\frac{1}{|C_1||C_2|} \sum_{a \in C_1} \sum_{b \in C_2} d(a,b).$$

dove C_1 e C_2 rappresentano i due cluster da unire e d la metrica prescelta.

Gli algoritmi più famosi appartenenti questa categoria sono SLINK (single-linkage) e CLINK (complete-linkage) [3].

1.3 Clustering spectral

Questa tipologia di clustering nasce dal bisogno di eliminare delle mancanze presenti negli altri metodi: i metodi partizionali riescono a creare cluster solamente di forma sferica, basandosi, come abbiamo già visto, sul concetto di centro; al contrario i metodi density-based hanno un approccio troppo sensibile ai parametri dati.

Per risolvere questi problemi si utilizza un grafo di similarità che ha come vertici le componenti da clusterizzare e, come valore degli archi, la similarità delle componenti tra cui questi sono sottesi.

Una volta creato il grafo, si procede con una serie di tagli minimi che possono essere trovati con un taglio normalizzato secondo la seguente formula:

$$NormCut(C_1, C_2) = \frac{Cut(C_1, C_2)}{Vol(C_1)} + \frac{Cut(C_1, C_2)}{Vol(C_2)}$$

dove C_1, C_2 sono due gruppi di nodi, w_{ij} il peso dell'arco sotteso tra i e j, Cut e Vol definiti come segue:

$$Cut(C_1, C_2) = \sum_{i \in C_1, j \in C_2} w_{ij}$$

$$Vol(C) = \sum_{i \in C, j \in V} w_{ij}$$

dove $Cut(C_1, C_2)$ calcola il peso totale delle connessioni tra C_1 e C_2 e $Vol(C)$ calcola il peso totale degli archi originati dal cluster C.

Uno degli algoritmi più famosi appartenente a questa categoria è Spectral [4].

1.4 Altri modelli di clutering

Come già detto, non esiste un solo modo, o algoritmo che sia, per riuscire a raggruppare degli oggetti. Qualsiasi intuizione potrebbe infatti generare un nuovo modello di pensiero, come successo per il *density-based* e per il *distribution-based*:

- Density-based: Negli algoritmi di clustering density-based, il raggruppamento avviene analizzando l'intorno di ogni punto dello spazio, connettendo regioni di punti con densità sufficientemente alta ed eliminando il rumore, ovvero gli elementi appartenenti a regioni con bassa densità [5].
  L'algoritmo più famoso appartenente a questa categoria è senz'altro DBSCAN.

- Distribution-based: Questa tipologia di algoritmi è quella che si avvicina di più allo studio statistico. I cluster possono essere definiti come insiemi di oggetti che appartengono, probabilmente, alla stessa distribuzione [6].
  L'algoritmo che definisce al meglio questa tipologia è il Gaussian Mixture Models.

Capitolo 2

Clusterify

Clusterify è un'applicazione web che offre agli utenti la possibilità di leggere gli stessi tweet che leggerebbero sul loro *newsfeed*, organizzati, però, in insiemi definiti dinamicamente in modo da permettere loro di soffermarsi solamente su testi appartenenti a categorie che questi reputano interessanti. L'idea di quest'applicazione è nata dal bisogno di non voler leggere tutti i tweet che una determinata persona pubblica.

Supponiamo che un utente X, aspirante cuoco, segua Y, cuoca neomamma conosciuta a livello nazionale. Quest'ultima pubblica due tweet: *1*) #Plumcake alla #Nutella: continuano gli esperimenti @GialloZafferano ;) *2*) Giocare una battaglia a #palledineve e perdere con il proprio piccolo di 3 anni <3 . Probabilmente X sarà molto interessato alla ricetta del *Plumcake* ma, al contempo, non attratto dalla vita personale di Y.

Clusterify dividerà questi tweet in gruppi distinti ed X avrà la possibilità di leggere solo quelli riferiti ad ambiti culinari senza essere disturbato da storie di vita personale, recensioni di film e quant'altro.

2.1 Background

Clusterify nasce con lo scopo di raggruppare per argomenti i tweet presenti nel newsfeed di un qualsiasi utente. Per questo scopo è necessario attribuire ad ogni tweet gli argomenti di cui tratta raggruppandoli per quanto questi siano collegati tra di loro.

Si è deciso di utilizzare la *Twitter API* per ottenere dati da twitter e *dataTXT* per estrarre gli argomenti e calcolarne le varie relazioni.

2.1.1 Twitter API

Twitter Inc. è un social network che permette ai propri utenti di scrivere e di leggere *micropost*, messaggi lunghi al più 140 caratteri, comunemente chiamati *tweet*.

Questa piattaforma, creata nel 2006 da Jack Dorsey, Evan Williams, Biz Stone e Noah Glass, ha spopolato fino a raggiungere nel 2013 più di 200 milioni di utenti attivi e più di 400 milioni di visite al giorno [7].

Ad oggi, Twitter è l'undicesimo sito internet visitato quotidianamente, secondo nei social network, preceduto solamente da Facebook [8].

Twitter possiede, come la maggior parte dei social network, un servizio interrogabile via API per ottenere e per caricare dati: nel primo caso si possono richiedere i tweet che compongono il *newsfeed* di un dato utente, i messaggi privati che due utenti si scambiano ed ancora alcuni dati relativi al profilo, come nome e cognome, nickname, immagine utente e così via; nel secondo caso, invece, si può pubblicare un tweet come se l'utente lo facesse dall'interfaccia web oppure modificare dati che Twitter ha salvato nel proprio database.

In entrambi i casi serve che l'utente interessato "firmi" un consenso che permetterà al programmatore di interagire con questi dati.

2.1.1.1 REST API v1.1

Il modo che Twitter offre per interagire con i propri dati è la REST API.

Con REST (Representational State Transfer) si indica un tipo di architettura software basato sull'idea di utilizzare la comunicazione tra macchine per mezzo di richieste HTTP.

Le applicazioni basate su questo tipo di architettura vengono nominate RESTful ed utilizzano, appunto, HTTP per tutte le operazioni di *CRUD*: Create, Read, Update e Remove.

Twitter mette a disposizione, nella documentazione per i programmatori [9], una lunga lista di possibili richieste – `https://dev.twitter.com/docs/api/1.1` – e per ognuna di queste una pagina dedicata dove vengono elencati i dati richiesti in input, i possibili filtri applicabili, la struttura dell'output che questa richiesta genererà ed infine un esempio richiesta/risposta per chiarire le idee.

OAuth, il protocollo di autenticazione che Twitter utilizza [10], permette agli utenti di approvare che un'applicazione agisca al loro posto, senza il bisogno di condividere i dati sensibili, quali username e password. OAuth scambia dei token per evitare il passaggio di questa tipologia di dati.

Gli sviluppatori possono così pubblicare e interagire con dati protetti, come ad esempio i service provider e al contempo proteggono le credenziali dei loro utenti [11].

2.1.2 dataTXT

DataTXT è un insieme di API semantiche sviluppate da SpazioDati S.r.l. `http://spaziodati.eu` che mira a estrarre significato da testi scritti in diverse lingue. Questo prodotto si diversifica dagli altri dello stesso genere in

quanto è stato pensato e ottimizzato per lavorare su testi molto corti, come i tweet.

DataTXT può estrarre da un testo delle *entità*, può categorizzare documenti in categorie stabilite dall'utente stesso e molto altro.

2.1.2.1 dataTXT NEX

DataTXT NEX (*Named Entity eXtraction*) è un API che permette di localizzazione e di estratte da un testo entità quali persone, organizzazioni, luoghi, espressioni di tempo, ecc.

Per ottenere queste informazioni basta richiedere via HTTP l'elaborazione dei dati prefissati in input. Segue un esempio: `https://api.dandelion.eu/datatxt/nex/v1/?lang=en&text=The%20doctor%20says%20an%20apple%20is%20better%20than%20an%20orange&include=type,abstract,categories,lod&$app_id=YOUR_APP_ID&$app_key=YOUR_APP_KEY`

Dopo l'elaborazione di una qualsiasi richiesta, una risposta viene consegnata.

Come tutte le chiamate HTTP, anche le risposte di dataTXT sono composte da un *header* e da un *body*.

Nella prima sezione sono presenti molte informazioni utili quali il numero di crediti utilizzati, quelli rimanente e la data in cui questi torneranno al valore iniziale.

```
Connection: keep-alive
Content-Length: 2748
Content-Type: application/json;charset=UTF-8
Date: Wed, 21 Oct 2015 16:29:37 GMT
Server: Apache-Coyote/1.1
X-DL-units: 1
X-DL-units-left: 999
X-DL-units-reset: 2015-10-22 00:00:00 +0000
```

Nella seconda, invece, sono presenti i dati veri e propri. *Annotations* è la lista composta dagli elementi trovati, con vari dettagli; questi elementi sono filtrati da alcuni parametri che possono essere passati nella richiesta, ad esempio la *min_confidence*.

```
{
  "timestamp": "2015-10-21T16:29:37",
  "time": 2,
  "lang": "en",
  "annotations": [{
      "...": "...",
  }, {
    "abstract": "The apple is the fruit of the apple tree...",
    "id": 18978754,
```

```
    "title": "Apple",
    "start": 19,
    "categories": [
      "Apples",
      "Malus",
      "Plants described in 1803",
      "Sequenced genomes"
    ],
    "lod": {
      "wikipedia": "http://en.wikipedia.org/wiki/Apple",
      "dbpedia": "http://dbpedia.org/resource/Apple"
    },
    "label": "Apple",
    "types": [
      "http://dbpedia.org/ontology/Eukaryote",
      "http://dbpedia.org/ontology/Plant",
      "http://dbpedia.org/ontology/Species"
    ],
    "confidence": 0.7869,
    "uri": "http://en.wikipedia.org/wiki/Apple",
    "end": 24,
    "spot": "apple"
  }, {
    "...": "...",
  }]
}
```


2.1.2.2 dataTXT REL

DataTXT REL è un servizio che, ad oggi, non è ancora presente sul mercato, ma che permetterà di capire quando un topic t_a è correlato ad un topic t_b, ricavando questo valore dalla seguente formula:

$$rel(t_a, t_b) = \frac{log(|in(t_a)|) - log(|in(t_a) \cap in(t_b)|)}{log(W) - log(|in(t_b)|)}$$

Come dataTXT-NEX, anche dataTXT-REL sarà interrogabile nello stesso modo: `https://api.dandelion.eu/datatxt/rel/v1/?topic1=Roma&topic2=Trento&lang=it&$app_id=YOUR_APP_ID&$app_key=YOUR_APP_KEY`.

L'*header* della risposta sarà uguale a quello di dataTXT-NEX, mentre il *body* sarà strutturato come segue:


```
{
  "time": 2,
  "relatedness": [{
```

```
    "weight": 0.7127059,
      "topic1": {
        "input": "Roma",
        "topic": {
          "id": 1209297,
          'title": "Roma",
          "uri": "http://it.wikipedia.org/wiki/Roma",
          "labe"l: "Roma"
        }
      },
      "topic2": {
        "input": "Trento",
        "topic": {
          "id": 100137,
          "title": "Trento",
          "uri": "http://it.wikipedia.org/wiki/Trento",
          "label": "Trento"
        }
      }
  }],
  "lang": "it",
  "timestamp": "2014-05-14T21:41:58.171"
}
```


2.2 Backend

Clusterify è un'applicazione Django – `https://www.djangoproject.com/` – pensata e creata basandosi sulla necessità di essere estesa e modellata secondo le preferenze degli sviluppatori.

Si può vedere il tutto come una struttura a mattoncini dove si possono scambiare dei blocchi per raggiungere il risultato atteso, senza necessariamente modificare il codice del *core*. Questi blocchi sono i *reader*, i *livelli d cache* e gli *algoritmi di cluster*.

Permettere di ottenere testi da più fonti significa creare altrettanti mezzi per interfacciare Clusterify a queste sorgenti, creando una *black-box* che trasforma i testi in input in un dato strutturato precedentemente definito. Clusterify nasce con TwitterReader che, come da nome, permette di leggere i dati direttamente dalle API di Twitter. Per creare un nuovo reader, quindi, basta soddisfare queste due regole.

La cache, invece, è una componente strutturale che viene utilizzata da tutto il progetto per evitare il replicarsi di richieste HTTP già fatte. Clusterify nasce con Redis – `http://redis.io/` – integrato, permettendo comunque di cambiare questa componente se lo sviluppatore lo desidera.

L'ultima componente estendibile sono gli algoritmi di clustering.

2.2.1 Algoritmi di clustering

Un algoritmo di clustering permette di raggruppare oggetti all'interno di insiemi, mantenendo uniti gli elementi che secondo alcune regole risultano più coesi e dividendo quelli meno simili.

Poiché non esiste un algoritmo ottimo per ogni caso pensabile, Clusterify ne comprende *quattro* nel core. Questi sono utili sia a rilasciare un sistema in grado di coprire la maggior parte dei casi d'uso, sia per valutare qual'è la performance migliore con l'input in questione.

I quattro algoritmi presenti in Clusterify sono presi da scikit-learn – `http://scikit-learn.org/`. Questa decisione è stata presa sia per offrire più di un algoritmo alla community, sia per valutare se la mia ipotesi iniziale circa il miglior algoritmo fosse corretta.

Questi algoritmi sono *K-means*, *Spectral*, *Affinity Propagation* e *Ward*.

2.2.1.1 K-means

K-means punta a creare K gruppi distinti di uguale varianza, con K, ovvero il numero di cluster richiesti, definito a priori.

L'algoritmo si può riassumere in questi punti:

Algorithm 1 K-means

1: Seleziona K punti come centroidi.
2: **repeat**
3: Forma K cluster assegnando ogni elemento al centroide più vicino.
4: Ricalcola i centroidi per ogni cluster.
5: **until** I centroidi non cambiano posizione (con possibile errore $\sim 1\%$).

Solitamente, la scelta iniziale della posizione dei K centroidi viene presa in modo casuale. In alcuni casi è conveniente utilizzare altre tecniche: ad esempio, se il numero di elementi si aggira attorno ad alcune centinaia di punti e se K è relativamente piccolo rispetto alla dimensione del cluster, si può utilizzare il clustering gerarchico fino a definire K cluster ed utilizzare i centroidi di questi cluster come i centroidi dell'intero sistema.

Tuttavia, per assegnare ogni elemento ad un centroide si ha bisogno di una misura in grado di quantificare il concetto di "vicinanza". Considerando che esistono diverse tipologie di dato che ipoteticamente si potrebbero clusterizzare, si ha bisogno di diverse funzioni per calcolare queste vicinanze: la distanza euclidea, conosciuta come $L2$, è solitamente utilizzata per dati rappresentabili su spazio euclideo; per la distanza tra documenti, invece, ha più senso utilizzare il *coseno di similitudine*.

Nel caso di dati rappresentabili su spazio euclideo misuriamo la qualità di un cluster utilizzando *SSE*, ovvero lo scarto quadratico medio. Il compito dell'algoritmo è di minimizzare quest'errore. Possiamo definire SSE come segue:

$$SSE = \sum_{i=1}^{K} \sum_{x \in C_i} dist(c_i, x)^2$$

dove *dist* è la distanza euclidea, C_i è l'i-esimo cluster e c_i è l'i-esimo centroide.

Si può dimostrare che il centroide che minimizza lo scarto quadratico medio del cluster è ottimale. Il centroide dell'i-esimo cluster è definito come segue:

$$c_i = \frac{1}{m_i} \sum_{x \in C_i} x$$

dove m_i è il numero di oggetti nell'i-esimo cluster.

Nel caso di documenti, invece, si deve massimizzare la similarità tra i documenti presenti in un cluster; questo valore viene detto *coesione* del cluster, l'analogo dello scarto quadratico medio, e viene calcolato come segue:

$$Total\ Cohesion = \sum_{i=1}^{K} \sum_{x \in C_i} cosine(x, c_i)$$

dove *cosine* è il *coseno di similitudine*.

In ogni caso l'algoritmo termina quando i centroidi assumono uno stato di fermo [12].

2.2.1.2 Spectral

Spectral è il nome di un algoritmo di clustering che punta ad utilizzare gli autovalori della matrice delle adiacenze per capire come partizionare gli elementi dell'insieme.

Dato un grafo, rappresentato dalla matrice delle adiacenze, l'algoritmo cerca il taglio minimo all'interno del grafo ed elimina gli archi che passano dal primo insieme al secondo. Questo procedimento continua in modo iterativo finche non si raggiungono, come nel caso del K-means, K cluster.

Lo spectral basa il suo funzionamento sulla matrice A della adiacenze, simmetrica per definizione, con $A_{ij} \geq 0$ rappresentante la similarità tra l'elemento in indice i e quello in indice j; questa matrice, conosciuta con il nome di matrice normalizzata di Laplace, viene così definita:

$$L^{norm} = I - D^{-1/2} A D^{-1/2},$$

dove D è la matrice diagonale costruita in questo modo:

$$D_{ii} = \sum_{j} A_{ij}.$$

L'algoritmo partiziona gli elementi in due insiemi distinti (B_1, B_2) basando la scelta del taglio sui valori dell'autovettore v relativo al secondo più piccolo autovalore calcolato su L^{norm} [?].

2.2.1.3 Affinity Propagation

Affinity Propagation è un algoritmo di clustering basato sul concetto di passaggio di messaggi tra le varie componenti. Una caratteristica fondamentale di quest'algoritmo è la mancanza di necessità di determinare a priori la quantità di cluster che saranno necessari.

Un dataset è descritto utilizzando degli esemplari identificati come le componenti più rappresentative dell'insieme. I messaggi trasmessi tra coppie sono utilizzati per calcolare l'idoneità che una componente ha di essere l'esemplare dell'altra, la quale sarà successivamente aggiornata in risposta al valore delle altre coppie. Questo processo viene ripetuto iterativamente finché il tutto non converge, dando vita al cluster.

Sia $(x_1, ..., x_n)$ l'insieme dei punti da clusterizzare e sia S una funzione che permette di definire la similarità di due punti.

L'algoritmo procede alternando due fasi di passaggio di messaggi, aggiornando due matrici:

- La matrice di responsabilità R ha valori $r(i,k)$ che quantificano quanto x_k sia ben situato nel cluster contenente x_i, prendendo in considerazione tutti gli altri elementi contenuti in questo insieme;
- La matrice di disponibilità A contiene i valori $a(i,k)$ che rappresentano quanto sia appropriato per x_i scegliere x_k come prossimo elemento del cluster a cui lui stesso appartiene.

Entrambe queste matrici sono inizializzate a 0 e, successivamente, saranno aggiornate in modo iterativo con le seguenti espressioni:

$$r(i,k) = s(i,k) - \max_{k' \neq k} \left\{ a(i,k') + s(i,k') \right\}$$

$$a(i,k) = \min \left(0, r(k,k) + \sum_{i' \notin \{i,k\}} \max(0, r(i',k)) \right) for i \neq k$$

$$a(k,k) = \sum_{i' \neq k} \max(0, r(i',k))$$

Il processo terminerà quando il tutto raggiungerà, come nel K-means, uno stato di fermo [13].

2.2.1.4 Ward

Il metodo Ward si basa sul clustering gerarchico di tipo agglomerativo. Ad ogni passo dell'esecuzione si unisce la coppia di cluster che possiede la distanza minima.

Questo metodo viene implementato dall'algoritmo Lance–Williams. Questo utilizza un approccio ricorsivo per aggiornare ad ogni passo la distanza tra i cluster, unendo i due gruppi più vicini.

Poiché non esiste un unico algoritmo basato su Ward, si può definire una famiglia di algoritmi, basati sullo stesso metodo, per calcolare la distanza tra due cluster. Sia

$$d_{(ij)k} = \alpha_i d_{ik} + \alpha_j d_{jk} + \beta d_{ij} + \gamma |d_{ik} - d_{jk}|,$$

la funzione generica che permette di calcolare questa distanza tra due cluster. Sia d_{ij} la distanza tra l'i-esimo e il j-esimo cluster; $d_{(ij)k}$ la distanza tra $C_i \cup C_j$ e C_k; e siano α_i, α_j, β e γ i parametri caratterizzanti gli algoritmi che compongono questa famiglia.

Ad esempio l'algoritmo che viene utilizzato da Clusterify definisce

$$\alpha_i = \alpha_j = \frac{n_l + n_k}{n_i + n_j + n_k}$$

$$\beta = \frac{-n_k}{n_i + n_j + n_k}$$

$$\gamma = 0$$

ed è conosciuto con il nome di *Ward's minimum variance method* [14].

2.3 Scelta dell'algoritmo di clustering

Una volta definiti i quattro algoritmi è necessario capire quale performa meglio nel clusterizzare argomenti secondo la relatedness offerta da dataTXT.

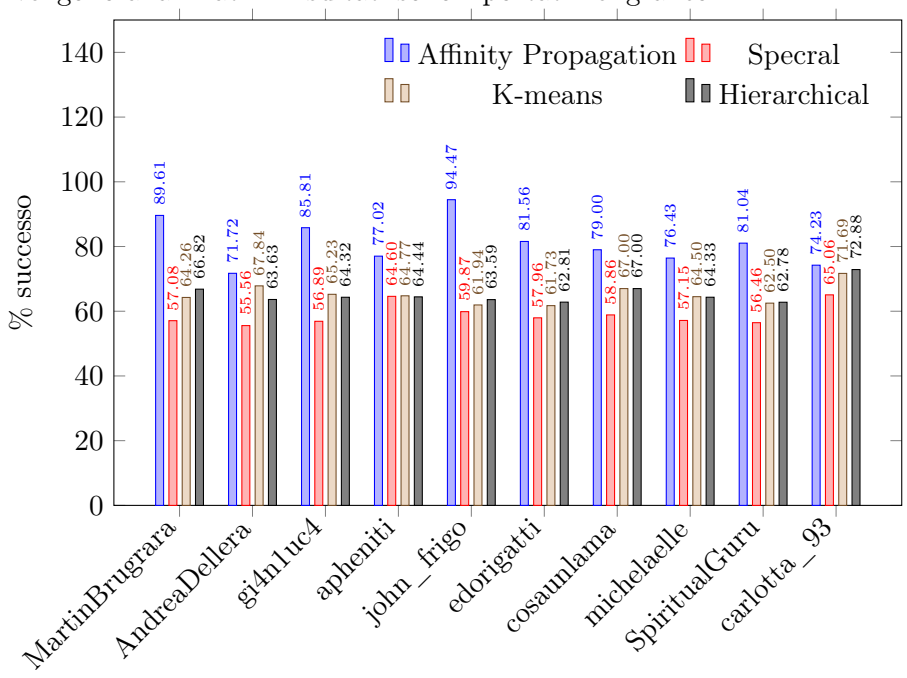
È stato chiesto a dieci persone di creare dei cluster partendo dalle entità estratte dagli ultimi 50 tweet che popolavano il loro newsfeed e il loro prodotto è stato paragonato con l'output di tutti gli algoritmi proposti.

Per questo compito è stata utilizzata la funzione "adjusted_rand_score" offerta da scikit-learn – `http://scikit-learn.org/stable/modules/generated/sklearn.metrics.adjusted_rand_score.html` – che permette di calcolare la similarità tra i cluster proposti e quello risultante. Per fare questo si considerano tutte le coppie di elementi e si contano quelle che sono state assegnate allo stesso o ad un altro cluster.

Per permettere agli intervistati di formare questi cluster è stato creato un apposito file su google drive – `https://docs.google.com/spreadsheets/d/1RNte6_I-cO5A23dKoaAyIRX5iWOGAEN3S25VIpXQLWU/edit?usp=sharing` – con

gli argomenti da clusterizzare pre-caricati ed è stato chiesto loro di segnare con lo stesso numero quelli che secondo loro avrebbero dovuto appartenere allo stesso cluster.

Lanciando `python manage.py analyze` dalla *root* di Clusterify questi dati vengono analizzati. I risultati sono riportati nel grafico:



Come si può dedurre dal grafico, ogni persona che ha creato il proprio cluster ideale si è avvicinata inconsciamente alla soluzione proposta dall'*Affinity Propagation*.

Una volta raccolti i dati è stato mostrato al campione intervistato il risultato proposto su di un nuovo dataset, generato sempre dall'account Twitter dei diretti interessati, ed il risultato nel 90% dei casi, era sensato ed utilizzabile.

Alla luce di questi risultati, Clusterify utilizzerà Affinity Propagation.

2.4 Workflow

Clusterify non è altro che un sistema di *pipe*, dove un comando viene eseguito prendendo come input l'output del precedente.

Per permettere che un elaborazione non ne blocchi un'altra, si è deciso di utilizzare celery – `http://www.celeryproject.org/` –, un sistema di code per la gestione di *task*. Così facendo, quando un utente richiede un'elaborazione, la si mette in coda e si informa l'utente che questa partirà appena possibile. Una volta terminata, l'utente riceverà una mail con la notifica di fine workflow.

Il workflow si può suddividere in tre parti:

1. Acquisizione dei testi
2. Annotazione dei testi
3. Clustering dei testi

2.4.1 Acquisizione dei testi

La componente di acquisizione testi, come detto precedentemente, è una delle parti pluggabili di Clusterify.

Questa fase si occupa del prendere i dati da una qualsivoglia fonte e di salvarli in modo conforme allo standard imposto. All'interno di questa componente è possibile implementare, se necessario, un livello di cache in modo da impedire di richiedere più volte le stesse informazioni.

In Clusterify, *TwitterReader* utilizza *Twython* – `https://github.com/ryanmcgrath/twython` –, una libreria Python che offre un accesso facile ai dati presenti su Twitter. In altre parole, Twython offre un *wrapper* alle API di Twitter, permettendo al programmatore di ottenere dati senza preoccuparsi della gestione errori, dell'aggiornamento delle API e quant'altro.

2.4.2 Annotazioni dei testi

La seconda fase del workflow si occupa di annotare i testi per mezzo di dataTXT.

Quest'operazione è fondamentale per un buon risultato finale in quanto si può incorrere nel riconoscimento di entità sbagliate; oppure, nel caso contrario, ci si può imbattere nell'estrazione di poche entità ma molto precise. Avere troppe annotazioni sbagliate può condurre all'avere dei dati non veritieri; il contrario potrebbe portare a non avere nemmeno un'entità estratta per testo analizzato e quindi, in fase di clustering, i testi analizzati da cui non si estraggono entità, verrebbero persi.

Al programmatore è quindi chiesto di scegliere molto attentamente i parametri che verranno passati all'estrattore, per non finire in nessuno dei due casi.

2.4.3 Clustering dei testi

L'ultima fase la si può pensare suddivisa a sua volta in due sotto fasi:

1. Creazione matrice delle adiacenze
2. Esecuzione dell'algoritmo di clustering

2.4.3.1 Creazione matrice di adiacenza

La maggior parte degli algoritmi di clustering necessita di un grafo pesato su cui operare. Questo è rappresentato per mezzo di una matrice delle adiacenze $N \times N$, dove N è la cardinalità dell'insieme composto da tutte le entità, e viene costruita in questo modo:

$$Adj(a,b) = \begin{cases} rel(a,b), & \text{se } a \neq b \\ 0, & altrimenti \end{cases}$$

ove Adj è la nostra matrice delle adiacenze, a e b sono i due topic e $rel(a,b)$ è la chiamata all'API dataTXT-REL che permette di capire quanto due entità siano correlate tra loro.

Si deduce che Adj è una matrice speculare, con valori reali compresi tra 0 e 1, e che presenta 0 sulla diagonale.

2.4.3.2 Esecuzione dell'algoritmo di clustering

Una volta generata la matrice delle adiacenze si può eseguire l'algoritmo di clustering precedentemente scelto. Dato l'output, i dati saranno processati nuovamente attraverso una funzione votata a due principali funzioni:

- Rendere l'output conforme con il modello prestabilito
- Aggiungere informazioni ad ogni componente del cluster

La prima si occuperà di trasformare la struttura dei dati generata dall'algoritmo in quella stabilita da Clusterify; la seconda, invece, si occuperà di aumentare l'informazione che questa contiene.

Ad esempio in Clusterify la seconda funzione aggiunge un dato che indica quanto una componente deve essere grande, calcolando questo dato sul numero di occorrenze che la stessa entità ha nel dataset.

2.5 Frontend

Ogni applicazione web è composta da due parti che collaborano per riuscire a offrire all'utente quello che ha richiesto: *backend* e *frontend*. Questa sezione parlerà di come è stato studiato e, successivamente, implementato il frontend.

2.5.1 Studio iniziale

Clusterify nasce per essere un'applicazione *user-centered*: si sono delineate tre *personas* in grado di rappresentare i differenti utilizzi che questa potesse supportare:

- Gigi Parigi *(studente di informatica - maschio - 20 anni)*: Gigi si è appena diplomato al liceo scientifico della sua città e si è immatricolato per seguire il corso in scienze informatiche. Utilizza Twitter sopratutto per rimare aggiornato sulle ultime notizie circa il mondo dell'open source. Si rende conto di leggere tweet di poco interesse quando gli utenti che segue parlano della propria vita privata.

- Rossella Secchi *(giornalista - femmina - 25 anni)*: Rossella è giornalista presso un emergente giornale online ed utilizza spesso Twitter per seguire il susseguirsi degli avvenimenti prima che questi vengano pubblicati da altri giornali. Il suo scopo è quello di riuscire a lanciare le notizie prima degli altri. Ritiene che avere le notizie raggruppate per argomento le salverebbe molto tempo speso in ricerca.

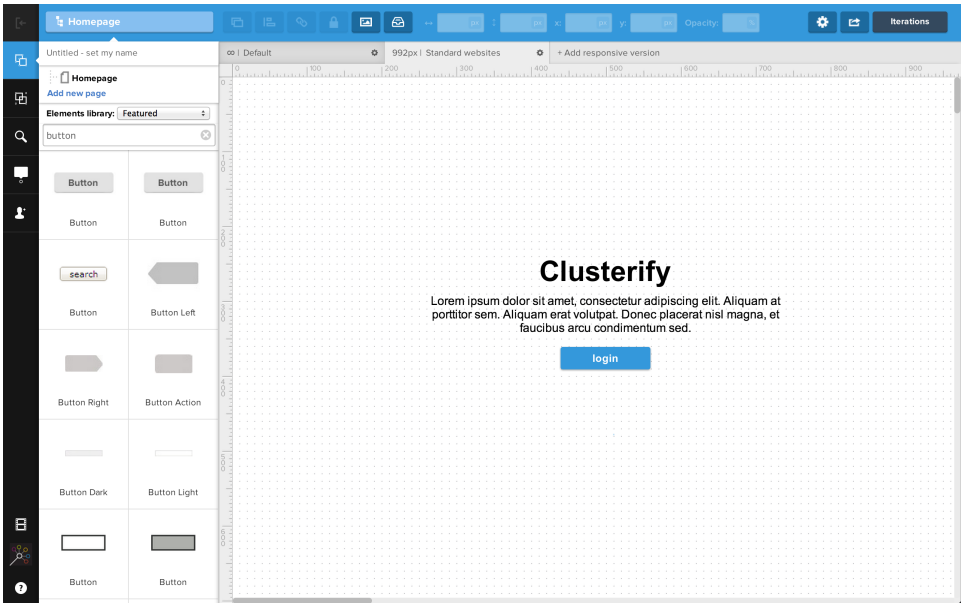
- Giovanni Fantuzzo *(fanatico di serie TV - maschio - 23 anni)*: Giovanni dopo aver smesso di guardare i cartoni animati ha capito che le serie TV sarebbero state un valido sostituto. Utilizza Twitter per seguire le news legate al campo: le interviste che fanno agli attori, nuovi rumors riguardanti nuove serie proposte e via dicendo. Sarebbe felice di poter avere tutte queste serie TV divise l'una dall'altra.

Tutte queste personas sono legate al mondo di Twitter: utilizzano questo social network ma in modi diversi. Si può dedurre che questa piattaforma deve essere facile da utilizzare e, soprattutto, accessibile anche ad utenti non molto legati al campo dell'informatica. È emerso il bisogno di mostrare una netta divisione tra i vari topic, rendendo le partizioni riconoscibili a colpo d'occhio. Infine, i gruppi devono permettere di essere esplorati portandoti ai tweet che lo hanno formato.

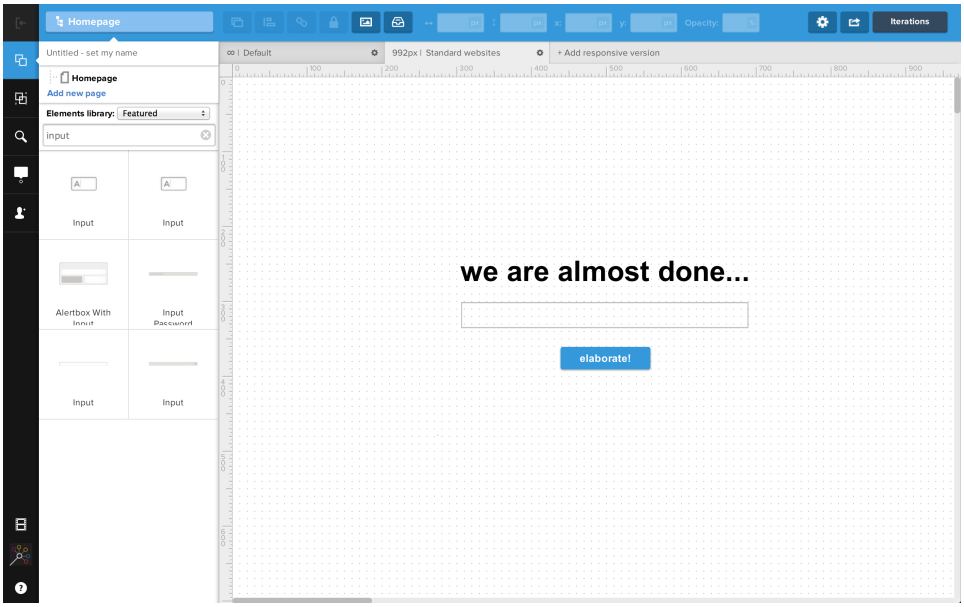
Sono stati abbozzati dei *mockup* pensando come queste personas si sarebbero comportate all'interno della piattaforma.

L'idea era quella di creare un *wizard* che permettesse in pochi *click* di ottenere il risultato finale:

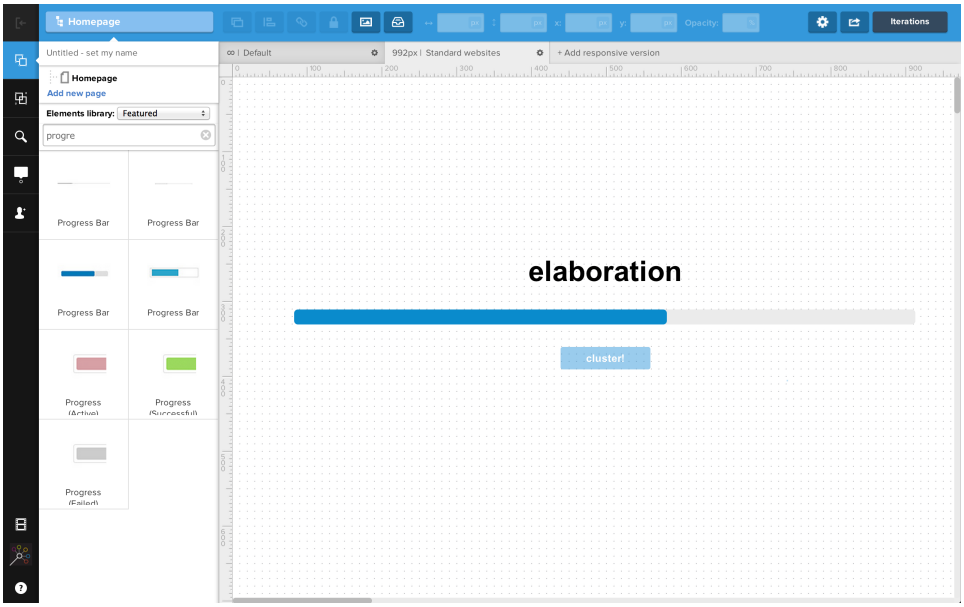
Pagina di benvenuto: pagina dove sarà possibile ottenere informazioni su Clusterify e dove sarà possibile loggarsi. Si è deciso di permettere di utilizzare l'account Twitter come unico modo per registrarsi/loggarsi con il fine di semplificare l'interfaccia.



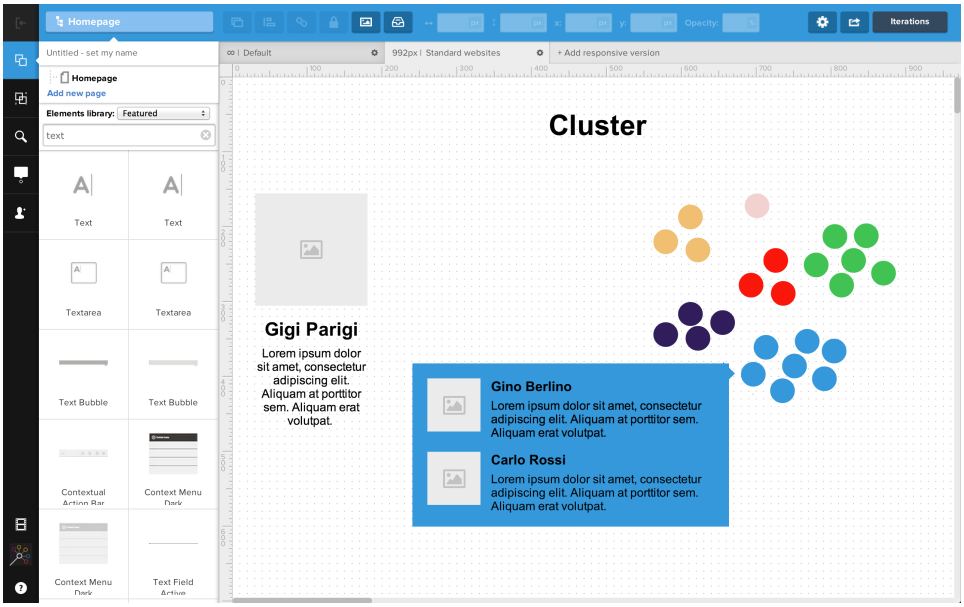
Inserimento email: pagina dove si richiederà di inserire la propria mail.



Stato dell'elaborazione: pagina che mostra all'utente lo stato dell'ultima elaborazione che questo ha lanciato.



Elaborazione: pagina che mostra il risultato dell'elaborazione.



Questi mockup sono stati proposti a cinque utenti che hanno permesso di capire alcuni problemi che erano stati introdotti.

- Nella pagina di richiesta dell'email non si capiva cosa venisse chiesto agli utenti. Si è decido di aggiungere degli aiuti visivi che permetteranno loro di capire.
- Nella pagina dedicata allo stato dell'elaborazione, gli intervistati non capivano cosa Clusterify stesse facendo, nemmeno che questo processo sarebbe continuato anche se loro avessero chiuso la pagina e che un'e-mail gli avrebbe notificati quando questo sarebbe finito. Si è deciso di aggiungere una scritta che chiarirà il comportamento di Clusterify.
- Nella pagina dedicata alla visualizzazione del cluster, viene dato poco spazio ai tweet mentre questi dovrebbero essere l'elemento fondamentale della piattaforma.
- Si è notata la necessità di avere una pagina con le elaborazioni passate.

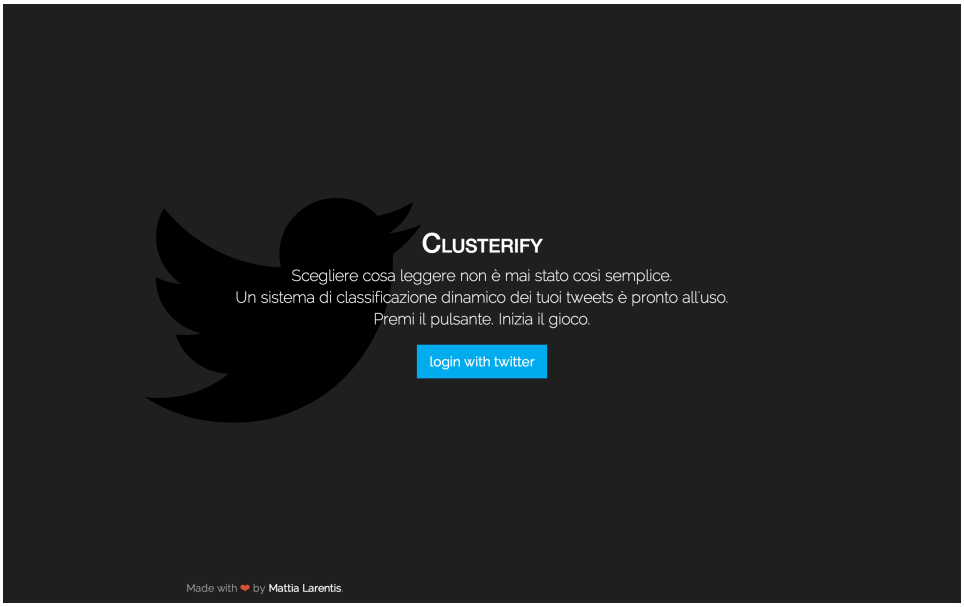
2.5.2 Studio finale

Clusterify utilizza come framework di base Bootstrap – `http://getbootstrap.com/` –, utile per alcune componenti quali la griglia ed i menù *dropdown*.

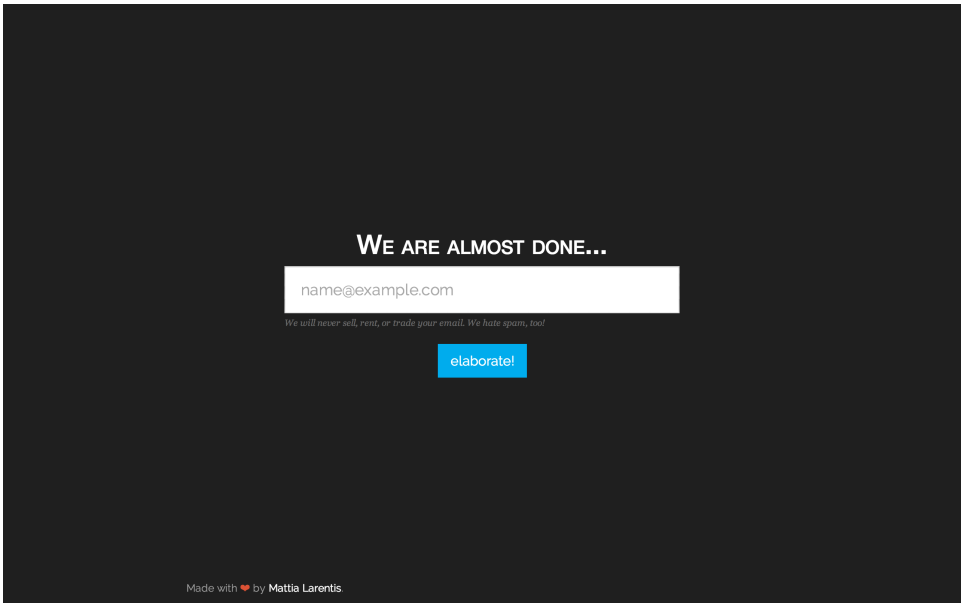
Per la parte di *data visualization* utilizzerà d3 – `http://d3js.org/` –, una libreria scritta in JavaScript utile per maneggiare dati e permettere la loro visualizzazione.

Dopo aver capito i problemi riscontrati nei mockup dai primi intervistati, si è implementato Clusterify.

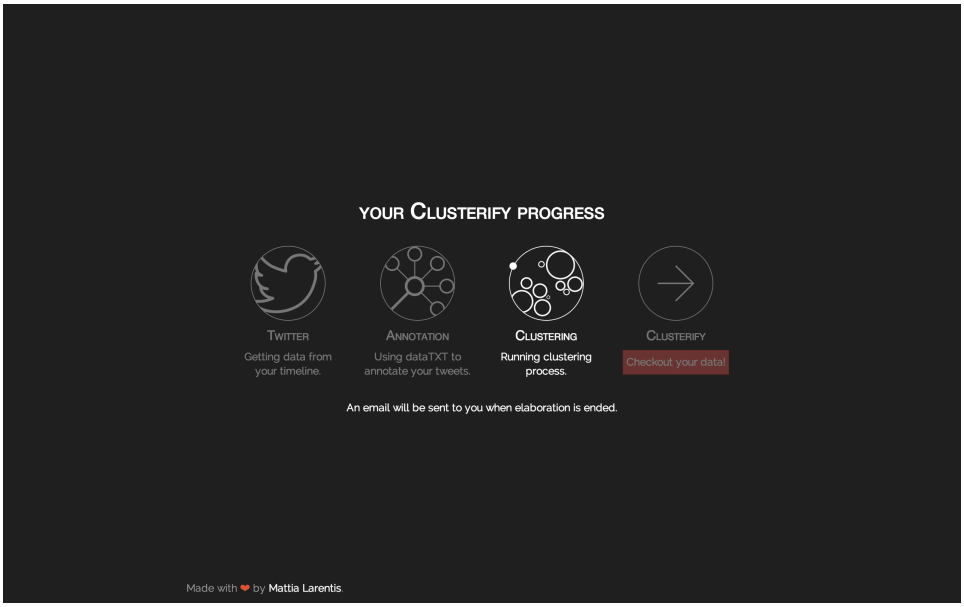
Pagina di benvenuto: landing page che offre la possibilità di registrarsi o di effettuare log in. Questa è anche la pagina dove si viene ridirezionati quando si richiede una visualizzazione di una pagina bloccata.



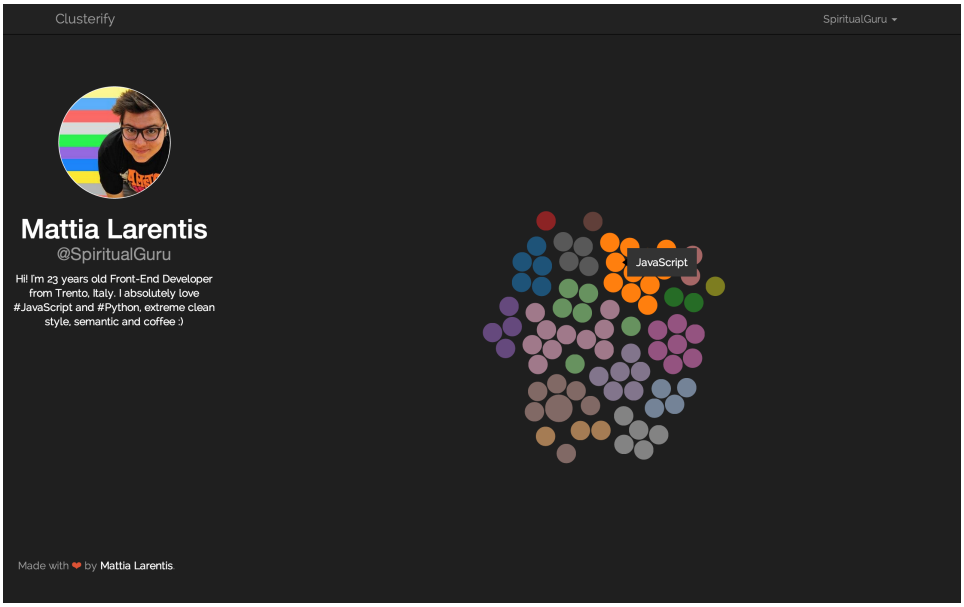
Inserimento email: nel momento in cui un utente si registra, viene domandato di inserire una propria email. L'indirizzo è necessario per contattare l'utente quando un'elaborazione termina.



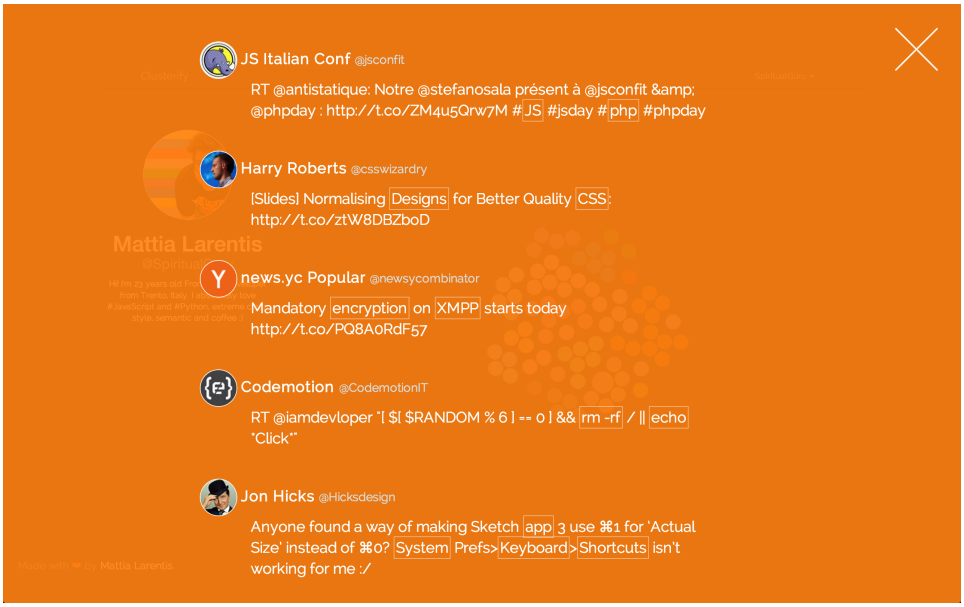
Stato dell'elaborazione: quando un utente lancia un'elaborazione viene mostrata questa *progress-bar*. Il processo assume la struttura del backend. Il passo illuminato è quello che sta venendo elaborato in quel momento.



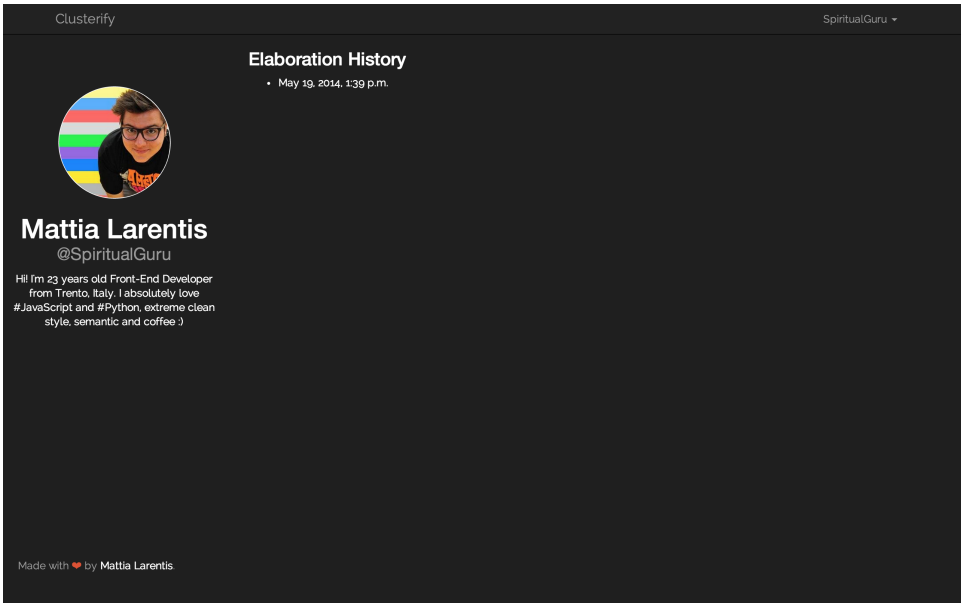
Elaborazione: pagina che mostra il cluster calcolato da Clusterify. Dal menù in alto è possibile lanciare una nuova elaborazione oppure visualizzare lo storico.



Tweet: da "Elaborazione" è possibile navigare ogni cluster clickando su un qualsiasi elemento che lo compone. Quando questo succede, una finestra modale si apre e permette di leggere tutti i tweet che sono stati raggruppati. Un tweet può appartenere a più cluster. Se un tweet viene premuto, si viene mandati alla pagina di Twitter dove il tweet risiede.



Storico delle elaborazioni: pagina dove sono visibili tutte le elaborazioni che un utente ha lanciato.



Successivamente, si è proposta la questa soluzione a 10 utenti, gli stessi che hanno valutato l'algoritmo di clustering, per capire se fossero sorti altri problemi nell'implementazione della piattaforma.

Tutti gli utenti sono rimasti soddisfatti ed il 60% ha detto che lo userebbe nella vita reale.

Conclusioni e sviluppi futuri

Clusterify è un prodotto pronto all'uso e già utilizzato da SpazioDati per pubblicizzare uno dei molteplici casi in cui dataTXT può essere utilizzato. Nato dall'idea di mostrare come la forza del clustering può essere utilizzata anche su testi molto corti, è stata soggetto di molti studi nel team di SpazioDati per poter creare un ulteriore prodotto vendibile al pubblico pagante.

Clusterify nasce però come *proof of concept* per mostrare queste potenzialità. In futuro potrebbe essere esteso con l'implementazione di un *workflow* totalmente dinamico, dove il processo di caricamento, annotazione e clustering dei testi possa essere di tipo incrementale. In questo scenario l'utente potrebbe abbandonare l'interfaccia tipica offerta da Twitter per avere il proprio stream aggiornato in tempo reale su Clusterify.

Questa composizione presenta tutti gli aspetti legati al *core* di Clusterify, i suoi possibili miglioramenti e le sue limitazioni. Si può anche affermare che è comunque un buon punto di partenza per nuovi programmatori che vorrebbero contribuire al progetto, portando nuove idee e nuova forza.

Penso che Clusterify sia uno strumento ben pensato per essere successivamente esteso e migliorato e che sia molto importante per i futuri clienti di SpazioDati; spero, infine, che questo progetto riesca a svilupparsi nel futuro prossimo.

Ringraziamenti

Anche se appare solamente il mio nome sulla copertina di questa tesi, questa non sarebbe mai esistita senza l'aiuto di molte persone: è un piacere e un dovere render loro grazie.

Ringrazio il mio supervisore Alberto Montresor per il suo incoraggiamento e il supporto fornitomi durante la stesura di questa composizione.

Un ringraziamento speciale va riservato a Stefano Parmesan ed a Federico Vaggi, i quali hanno prestato molta pazienza e dedizione nel seguirmi ed impegno nell'aiutarmi.

Ho acquisito molte conoscenze circa lo sviluppo web durante le mie attività lavorative e alla passione che i miei colleghi sono stati in grado di trasmettermi; un grazie al team di SpazioDati e ai trenta3dev per il loro appoggio e per i bei momenti vissuti con loro.

Ringrazio mamma e papà in quanto mamma e papà.

Un grazie è tutto per Michela e per Carlotta che mi hanno spinto a dare del mio meglio anche quest'anno.

Infine vorrei ringraziare Michele Pittoni e tutti i miei compagni di corso che mi hanno supportato in questi tre anni di studio.

Bibliografia

[1] Edoardo Sernesi. Geometria i.

[2] springerreference. Partitional clustering. http://www.springerreference.com/docs/html/chapterdbid/179343.html.

[3] Matteo Matteucci. Hierarchical clustering algorithms. http://home.deib.polimi.it/matteucc/Clustering/tutorial_html/hierarchical.html.

[4] Jitendra Malik Jianbo Shi. Normalized cuts and image segmentation.

[5] Hans-Peter Kriegel, Peer Kröger, Jörg Sander, and Arthur Zimek. Density-based clustering. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(3):231–240, 2011.

[6] Jia Li. Mixture models. http://sites.stat.psu.edu/~jiali/course/stat597e/notes2/mix.pdf.

[7] Karen Wickre. Celebrating twitter7. https://blog.twitter.com/2013/celebrating-twitter7.

[8] Alexa. twitter.com site overview. http://www.alexa.com/siteinfo/twitter.com.

[9] Twitter. Documentation | twitter developers. https://dev.twitter.com/docs.

[10] Twitter. Authentication & authorization | twitter developers. https://dev.twitter.com/docs/auth.

[11] Twitter. Oauth faq | twitter developers. https://dev.twitter.com/docs/auth/oauth/faq.

[12] Vipin Kumar Pang-Ning Tan, Michael Steinbach. Introduction to data mining. http://www-users.cs.umn.edu/~kumar/dmbook/index.php.

[13] Delbert Dueck Brendan J. Frey. Clustering by passing messages between data points.
http://www.sciencemag.org/content/315/5814/972.short.

[14] Pierre Legendre Fionn Murtagh. Ward's hierarchical clustering method: Clustering criterion and agglomerative algorithm.
http://arxiv.org/pdf/1111.6285v2.pdf.