Linguaggi di Programmazione
AA 2015-2016
Progetto settembre 2016
***k*-medie**

Marco Antoniotti e Gabriella Pasi
DISCo

21 agosto 2016

Scadenza

La consegna di questo elaborato è fissata per il 23 settembre 2016 alle ore 23:59 GMT+1.

Premessa

LEGGERE ATTENTAMENTE TUTTO IL TESTO!

1 Introduzione

Uno degli algoritmi principali (e più semplici) utilizzati nell'*analisi statistica dei dati*[1] è noto come l'algoritmo di *clustering non supervisionato* ("unsupervised") delle ***k*-medie**.

L'obiettivo di un algoritmo di clustering è, dato un insieme di n *oggetti* (o *osservazioni*), partizionarli in k sottoinsiemi (o categorie non predefinite) che raggruppino oggetti che condividono delle propriet. Ad esempio un algoritmo di clustering applicato a delle immagini telerilevate potrebbe partizionare le immagini sulla base della tipologia di scena rappresentata, quale centri abitati, boschi, superfici acquee, ecc. In particolare, l'algoritmo di clustering delle ***k*-medie** è di partizionare n *osservazioni* in k *clusters* (gruppi), dove ogni osservazione appartiene al gruppo in cui cade la *media* più "vicina". La "media" (detta *centroide*) serve come "prototipo" del gruppo. Il centroide che rappresenta una categoria viene in questo caso calcolato come la media degli oggetti del gruppo e ne costituisce il prototipo.

In generale il problema è NP-hard, ma la variante "euristica" di Lloyd dell'algoritmo ***k*-medie** è una soluzione abbastanza buona ed efficace. Una limitazione dell'algoritmo ***k*-medie** è che il parametro k deve essere specificato dall'utente in anticipo.

Il vostro compito è di construire una libreria Common Lisp ed una libreria Prolog che implementino l'algoritmo ***k*-medie** di Lloyd.

Per una descrizione dell'algoritmo delle ***k*-medie** potete guardare G. James, D. Witten, T. Hastie, R. Tibshirani, *An Introduction to Statistical Learning*, Springer, 2015, o al più avanzato T. Hastie, R. Tibshirani, J. Friedman, *The Elements of Statistical Learning, Data Mining, Inference, and Prediction*, Springer, 2009, oppure anche le descrizioni di Wikipedia (Inglese).

L'Algoritmo 1 rappresenta (in pseudo codice) i passi principali dell'algoritmo ***k*-medie**.

[1] Statistical Learning, Machine Learning, Data Analysis, Big Data, etc. etc. etc.

Algoritmo 1 *k*-medie di Lloyd: pseudo codice.

KM(n observations, k) $\rightarrow$ k clusters

1: cs $\leftarrow$ Initialize(k)
   Crea k centroidi iniziali, ad esempio usando il metodo di Forgy che sceglie casualmente k delle osservazioni iniziali.
2: clusters $\leftarrow$ {}
3: clusters′ $\leftarrow$ Partition(observations, cs)
   Raggruppa le "observations" attorno ai k centroidi in "cs".
4: **if** clusters = clusters′ **then**
5: **return** clusters
6: **else**
7: clusters $\leftarrow$ clusters′
8: cs $\leftarrow$ RecomputeCentroids(clusters)
   Ricalcola il "centroide" di ogni gruppo.
9: **goto** 3
10: **end if**

2 Requisiti Progetto

Innanzitutto il progetto è realizzabile in modo completamente funzionale (o logico). Non potete usare operazioni di assegnamento (set, setq, setf) o asserzioni sulla base dati Prolog, se non per casi assolutamente necessari e dopo aver ricevuto esplicito permesso.

Osservazioni

Le "osservazioni" sono, nel caso più semplice, dei vettori numerici. Sempre per semplicità[2] in Common Lisp ed in Prolog dovrete rappresentare le "osservazioni" con delle *liste*.

Dato che il cuore dell'algoritmo ***k*-medie** è costituito da un'operazione di calcolo di una distanza (Euclidea) tra vettori, dovrete implementare una serie di operazioni vettoriali.

- somma, sottrazione, prodotto scalare
- prodotto interno (euclideo)
- norma

Esempi Common Lisp

Creiamo un vettore `v3`

```
CL prompt> (defparameter v3 (list 1 2 3))
V3

CL prompt> v3
(1 2 3)
```

Ora calcoliamo la sua norma, ovvero...

```
CL prompt> (srqt (innerprod V3 V3))
3.7416575 ; Il risultato può variare.
```

[2] Ovviamente non per efficienza.

dove `innerprod` è il prodotto interno. Naturalmente possiamo anche avere

```
CL prompt> (norm V3)
3.7416575
```

Somme, etc...

```
CL prompt> (vsum V3 (list 10 0 42))
(11 2 45)
```

Le funzioni `map`, `mapc`, `mapcar`, `mapcan`, `reduce` etc., sono più che utili in questo frangente.

Esempi Prolog

Ricordiamoci un vettore...

```
?- new_vector(v3, [1, 2, 3]).
true

?- vector(v3, V).
V = [1, 2, 3]
```

Ora calcoliamo la sua norma...

```
?- vector(v3, V), innerprod(V, V, IP), N is sqrt(IP).
V = [1, 2, 3]
N = 3.7416575

?- vector(v3, V), norm(V, N).
V = [1, 2, 3]
N = 3.7416575

?- vector(v3, V), vsum(V, [10, 0, 42], S).
V = [1, 2, 3]
S = [11, 2, 45]
```

Interfaccia

Common Lisp

La libreria Common Lisp dovrà fornire una funzione `km` che costruisca la partizione dell'insieme di osservazioni in k gruppi (clusters). Altre funzioni di utilità sono elencate di seguito.

`km` *observations* $k \rightarrow$ *clusters*

Il parametro *observations* è una lista di vettori (ovvero liste), il parametro k è il numero di clusters da generare. Il risultato *clusters* è una lista di gruppi, ovvero di liste di vettori (che, ripetiamo, sono liste). La funzione deve fallire se il numero di osservazioni è minore di k.

`centroid` *observations* $\rightarrow$ *centroid*

La funzione `centroid` ritorna il centroide (i.e., la "media") dell'insieme di osservazioni *observations* (una lista di vettori, ovvero di altre liste).

Nota bene. Il centroide di un insieme di vettori non è necessariamente un elemento dell'insieme dato.

`vsum` *vector1 vector2* $\rightarrow v$

La funzione `vsum` calcola la somma (vettoriale) di due vettori.

`vsub` *vector1 vector2* $\rightarrow v$

La funzione `v` calcola la differenza (vettoriale) di due vettori.

`innerprod` *vector1 vector2* $\rightarrow v$

La funzione `innerprod` calcola il prodotto interno (vettoriale) di due vettori. Il valore ritornato v è uno scalare.

`norm` *vector* $\rightarrow v$

La funzione `norm` calcola la norma euclidea di un vettore. Il valore ritornato v è uno scalare.

Prolog

La libreria Prolog dovrà fornire una predicato `km` che costruisca la partizione dell'insieme di osservazioni in k gruppi (clusters). Altri predicati di utilità sono elencate di seguito.

`km`(*Observations*, *K*, *Clusters*)

Il parametro *Observations* è una lista di vettori (ovvero liste), il parametro K è il numero di clusters da generare. Il predicato `km/3` è vero quando *Clusters* è una lista di gruppi che corrisponde alla partizione di *Observations* in k clusters.
Il predicato `km/3` deve fallire se il numero di osservazioni è minore di K.

`centroid`(*Observations*, *Centroid*)

Il predicato `centroid/2` è vero quando *Centroid* è il centroide (i.e., la "media") dell'insieme di osservazioni *Observations* (una lista di vettori, ovvero di altre liste).
Nota bene. Il centroide di un insieme di vettori non è necessariamente un elemento dell'insieme dato.

`vsum`(*Vector1*, *Vector2*, *V*)

Il predicato `vsum/3` è vero quando V è la somma (vettoriale) di due vettori.

`vsub`(*Vector1*, *Vector2*, *V*)

Il predicato `vsub/3` è vero quando V è la sottrazione (vettoriale) del vettore *Vector2* da *Vector1*.

`innerprod`(*Vector1*, *Vector2*, *R*)

Il predicato `innerprod/3` è vero quando R è il prodotto interno (vettoriale) di due vettori. Il valore R è uno scalare.

`norm`(*Vector*, *N*)

Il predicato `norm/2` è vero quando N è la norma euclidea di un vettore. Il valore ritornato N è uno scalare.

`new_vector`(*Name*, *Vector*)

Il predicato `new_vector/2` è vero quando a *Name* (un atomo Prolog) viene associato un vettore *Vector*. In questo caso potete usare `assert`.

Note ed esempi

Considerate l'insieme di *osservazioni* (in 2D):

$$\begin{aligned}\mathcal{O} \quad = \quad & \{(3.0, 7.0), (0.5, 1.0), (0.8, 0.5), (1.0, 8.0),\\ & (0.9, 1.2), (6.0, 4.0), (7.0, 5.5),\\ & (4.0, 9.0), (9.0, 4.0)\}.\end{aligned}$$

Le tre clusters (con $k = 3$) calcolate dall'algoritmo ***k*-medie** sono:

1. $\{(1.0, 8.0), (3.0, 7.0), (4.0, 9.0)\}$,
2. $\{(0.5, 1.0), (0.8, 0.5), (0.9, 1.2)\}$,
3. $\{(6.0, 4.0), (7.0, 5.5), (9.0, 4.0)\}$.

La Figura 1 mostra la disposizione di ogni punto e di ogni cluster. Notate che i centroidi *non* fanno parte dell'insieme iniziale di osservazioni.

Esempi

Common Lisp

L'ordine in ognuno dei gruppi trovati non è importante.

```
CL prompt> (defparameter observations
             '((3.0 7.0) (0.5 1.0) (0.8 0.5) (1.0 8.0)
               (0.9 1.2) (6.0 4.0) (7.0 5.5)
               (4.0 9.0) (9.0 4.0)))
OBSERVATIONS
```

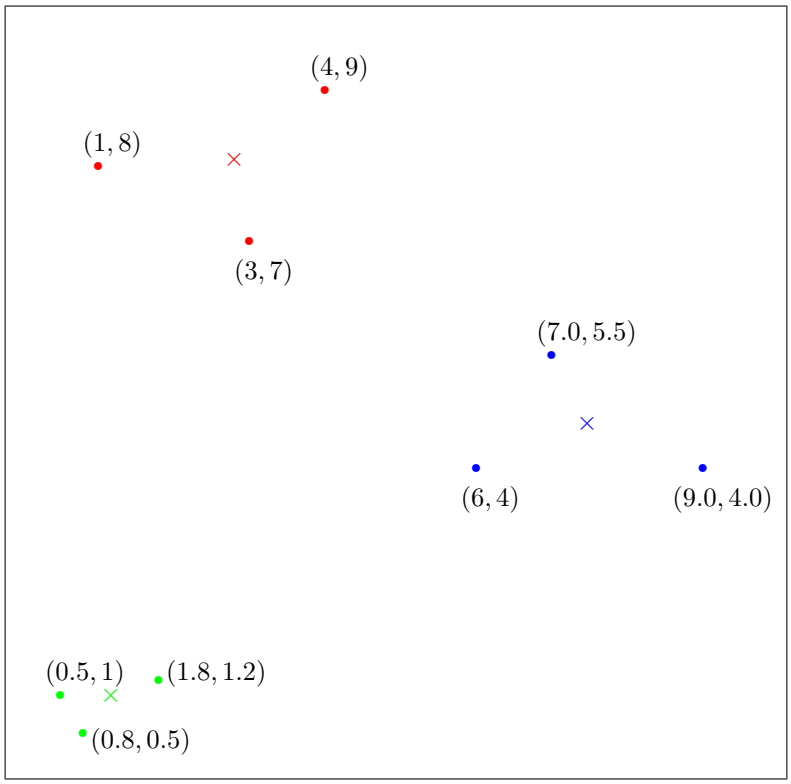

Fig. 1: Un esempio con tre clusters (colorate in rosso, verde e blu) di 9 osservazioni (in questo caso punti a 2D). I centroidi (marcati con '×') a $(2.666, 8)$, $(1.033, 0.9)$ and $(7.333, 4.5)$ *non* fanno parte dell'insieme iniziale di osservazioni.

```
CL prompt> (km observations 3)
(((1.0 8.0) (3.0 7.0) (4.0 9.0))
 ((0.5 1.0) (0.8 0.5) (0.9 1.2))
 ((6.0 4.0) (7.0 5.5) (9.0 4.0)))
```

Prolog

L'ordine in ognuno dei gruppi trovati non è importante.

```
?- km([[3.0, 7.0], [0.5, 1.0], [0.8, 0.5], [1.0, 8.0],
       [0.9, 1.2], [6.0, 4.0], [7.0, 5.5],
       [4.0, 9.0], [9.0, 4.0]],
      3,
      Clusters).
Clusters = [[[1.0, 8.0], [3.0, 7.0], [4.0, 9.0]],
            [[0.5, 1.0], [0.8, 0.5], [0.9, 1.2]]
            [[6.0, 4.0], [7.0, 5.5], [9.0, 4.0]]]
```

Note

L'algoritmo delle ***k*-medie** è, di fatto, una serie di cicli innestati. Ricordate che questa funzione:

```
(defun fatt (x)
  (let ((result 1))
    (loop while (> x 0)
          do (setf result (* x result)
                   x (1- x)))
    result))
```

È del tutto equivalente a questa:

```
(defun fatt (x &optional (result 1))
  (if (zerop x)
      result
      (fatt (1- x) (* x result))))
```

In altre parole, potete (dovete!) sostituire ogni ciclo con una funzione ricorsiva (in coda).

Everybody knows how to search the Internet!

...in altre parole, sono ben noti (quasi) tutti i pacchetti software che implementano (in Common Lisp o in Prolog) l'algoritmo ***k*-medie**.

3 Da consegnare...

LEGGERE ATTENTAMENTE LE ISTRUZIONI QUI SOTTO.

PRIMA DI CONSEGNARE, CONTROLLATE **ACCURATAMENTE** CHE TUTTO SIA NEL FORMATO CORRETTO E CON LA STRUTTURA DI CARTELLE RICHIESTA.

Dovete consegnare:

> Uno `.zip` file dal nome `<Cognome>_<Nome>_<matricola>_km_LP_201609.zip` *che conterrà una cartella dal nome* `<Cognome>_<Nome>_<matricola>_km_LP_201609`.

Cognomi e nomi multipli dovranno essere scritti sempre con in carattere "underscore" ('_'). Ad esempio, "Gian Giacomo Pier Carl Luca De Mascetti Vien Dal Mare" che ha matricola 424242 diventerà:
`De_Mascetti_Vien_Dal_Mare_Gian_Giacomo_Pier_Carl_Luca_424242_km_LP`
Inoltre...

- Nella cartella dovete avere una sottocartella di nome `Lisp` e una sottocartella di nome `Prolog`.
- Nella directory `Lisp` dovete avere:
  - un file dal nome `km.lisp` che contiene il codice di `km`, `vsum`, etc.
    - Le prime linee del file **devono essere dei commenti con il seguente formato**, ovvero devono fornire le necessarie informazioni secondo le regole sulla collaborazione pubblicate su Moodle.

      ```
      ;;;; <Cognome> <Nome> <Matricola>
      ;;;; <eventuali collaborazioni>
      ```

      Il contenuto del file deve essere ben commentato.
  - Un file `README` in cui si spiega come si possono usare le funzioni definite nella libreria.
- Nella directory `Prolog` dovete avere:
  - un file dal nome `km.pl` che contiene il codice di `km/3`, `vsum/3`, etc.
    - Le prime linee del file **devono essere dei commenti con il seguente formato**, ovvero devono fornire le necessarie informazioni secondo le regole sulla collaborazione pubblicate su Moodle.

      ```
      % <Cognome> <Nome> <Matricola>
      % <eventuali collaborazioni>
      ```

      Il contenuto del file deve essere ben commentato.
  - Un file `README` in cui si spiega come si possono usare i predicati definiti nel programma.

ATTENZIONE! Consegnate solo dei files e directories con nomi fatti come spiegato. Niente spazi extra **e soprattutto niente `.rar` or `.7z` o `.tgz` – solo `.zip`!**
Repetita juvant! *NON CONSEGNARE FILES* `.rar`!!!!

Esempio:
File `.zip`:

```
Antoniotti_Marco_424242_km_LP_201609.zip
```

Che contiene:

```
prompt$ unzip -l Antoniotti_Marco_424242_km_LP_201609.zip
Archive:  Antoniotti_Marco_424242_km_LP_201609.zip
  Length     Date   Time    Name
 --------    ----   ----    ----
        0  12-02-14 09:59   Antoniotti_Marco_424242_km_LP_201609/
        0  12-04-14 09:55   Antoniotti_Marco_424242_km_LP_201609/Lisp/
     4623  12-04-14 09:51   Antoniotti_Marco_424242_km_LP_201609/Lisp/km.lisp
    10598  12-04-14 09:53   Antoniotti_Marco_424242_km_LP_201609/Lisp/README.txt
        0  12-04-14 09:55   Antoniotti_Marco_424242_km_LP_201609/Prolog/
     4623  12-04-14 09:51   Antoniotti_Marco_424242_km_LP_201609/Prolog/km.lisp
    10598  12-04-14 09:53   Antoniotti_Marco_424242_km_LP_201609/Prolog/README.txt
 --------                   -------
    30442                   7 files
```

3.1 Valutazione

Il programma sarà valutato sulla base di una serie di test standard, oltre agli esempi riportati in questo testo.