

Introduction to ML

Marini Mattia

3^o semestre 3^o anno

Indice

1	Gradient descent	2
2	Unsupervised learning - riduzione dimensione	3
2.0.1	Media e varianza	3
2.0.2	Autovalori e autovettori	4
2.0.3	Obiettivo	6
2.0.4	Equivalenza metodi:	6
2.0.5	Pseudocodice	7
2.0.6	Trovare di quanto ridurre	8
3	Clustering	8
3.1	k-means	8
3.1.1	Pseudocodice	8
3.1.2	Note	9
3.2	EM-clustering	9
3.3	Spectral clustering	10
3.4	Clustering gerarchico	11
3.4.1	Pseudocodice	11
3.4.2	Vantaggi rispetto a k-means	12
4	Reti neurali	12
4.1	Calcolo output predetto	13
4.1.1	Classificazione: softmax + cross-entropy	13
4.1.2	Regressione: softmax + cross-entropy	14
5	Miscellanea nozioni	14
5.1	Cosine distance	14

1

Gradient descent

Per valutare la qualità di un modello, si usa una funzione di costo che misura la differenza fra i valori previsti e quelli reali. L'obiettivo è minimizzare questa funzione di costo.

Definizione 1: *Funzione di loss*

Una *funzione di loss* è una funzione che misura la differenza tra i valori previsti e quelli reali.

Ad esempio, per un modello di classificazione lineare, la funzione di loss può essere il numero di punti che stanno dalla parte sbagliata del piano

$$\text{loss}(w, b) = \sum_{i=1}^n \mathbf{1}[y_i(w \cdot x_i + b) < 0]$$

l'obiettivo è minimizzare questa funzione ma spesso è molto difficile (NP-hard in fact). Spesso quindi si cerca un'approssimazione, cercando i minimi locali in funzioni che limitano la funzione di loss dall'alto:

Definizione 2: *Funzione di loss surrogata*

Una *funzione di loss surrogata* è una funzione che approssima la funzione di loss, limitandola dall'alto.

Di seguito le seguenti funzioni di loss più comuni:

0-1 loss	$\ell_{0/1}(y, y') = \mathbf{1}[y y' \leq 0]$
hinge loss	$\ell_{\text{hinge}}(y, y') = \max(0, 1 - y y')$
exponential loss	$\ell_{\text{exp}}(y, y') = \exp(-y y')$
square loss	$\ell_{\text{sq}}(y, y') = (y - y')^2$

Per trovare il minimo di una funzione di loss surrogata posso usare il *gradient descent*, cercando di "scivolare" verso il minimo andando "in discesa" verso ogni asse singolarmente. Ad esempio utilizzando la loss function esponenziale

$$\text{loss}(w, b) = \sum_{i=1}^n \exp(y_i(w \cdot x_i + b))$$

in particolare se cerco di modificare un peso alla volta considerando la loss function in una sola delle sue k dimensioni, ho che:

$$\begin{aligned} w_j &= w_j - \eta \frac{d}{dw_j} \text{loss}(w, b) \\ &= w_j - \eta \sum_{i=1}^n -y_i x_{ij} \exp(-y_i(w \cdot x_i + b)) \end{aligned}$$

dunque abbiamo ottenuto la formula per aggiornare ciascun peso:

$$w_j = w_j + \eta \sum_{i=1}^n y_i x_{ij} \exp(-y_i(w \cdot x_i + b))$$

il che equivale a considerare ciascun punto nel training set e aggiornare il peso j -esimo come segue:

$$w_j = w_j + \eta y_i x_{ij} \exp(-y_i(w \cdot x_i + b)) \quad \forall i \in [0, \dots, n]$$

Algoritmo 1: *Gradient descent*

Input: Learning rates $\eta = [\eta_1, \dots, \eta_n]$, loss function L , # of iterations K

Output: Weights $w = [w_1, \dots, w_k]$

gradient_descent(X, Y, L, η):

```

     $w^{(0)} = [0, \dots, 0]$  ;           ▷ Dati come input in alternativa
    for  $k \in [0, \dots, K]$  do
         $g^{(k)} = \nabla_{w^{(k)}} L$  ;           ▷ shift for each direction
         $w^{(k)} = w^{(k-1)} - \eta^{(k)} g^{(k)}$  ;   ▷ move in direction of gradient
    return  $w^{(k)}$ 
```

Nota che $w^{(k)}$ indica i pesi relativi alla k -esima iterazione, ossia dopo essere stati migliorati k volte. Similmente $g^{(k)}$ è il vettore contenente le derivate parziali (ossia considerate una direzione alla volta) della loss function nel punto ottenuto dopo k iterazioni:

$$\nabla_{w^{(k)}} L = \begin{bmatrix} \frac{\partial L}{\partial w_1^{(k)}} & \frac{\partial L}{\partial w_2^{(k)}} & \dots & \frac{\partial L}{\partial w_N^{(k)}} \end{bmatrix}$$

dove L è la *funzione di loss*

2 Unsupervised learning - riduzione dimensione

L'idea generale è quella di "appiattire" i punti nell'asse parallela all'asse con varianza massima, così da ridurre al massimo la perdita di informazione

Formalmente si prende un versore w che parte da un punto c . Un versore significa un vettore con norma 1, ossia

$$|w| = 1 \Leftrightarrow w^T w = 1$$

dopodichè si condierano tutte le proiezioni dei punti x_i su questo versore, ossia

$$t_i = (x_i - c)^T w$$

2.0.1 Media e varianza

La media serve per il calcolo della varianza, la quale è il criterio secondo cui eseguiamo la riduzione di dimensione:

$$\begin{aligned} \mathbf{E}[t] &= \frac{1}{n} \sum_{i=1}^n t_i = \frac{1}{n} \sum_{i=1}^n (x_i - c)^T w \\ &= \bar{x}^T w - c^T w \end{aligned}$$

dove

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$$

La varianza indica quanto i valori sono sparsi lungo l'asse identificato dal vettore w :

$$\begin{aligned} \mathbf{Var}[t] &= \frac{1}{n} \sum_{i=1}^n (t_i - E[t])^2 \\ &= \frac{1}{n} \sum_{i=1}^n [x_i^\top w - c^\top w - \bar{x}^\top w + c^\top w]^2 \\ &= \frac{1}{n} \sum_{i=1}^n [(x_i - \bar{x})^\top w]^2 \\ &= \frac{1}{n} \sum_{i=1}^n (\bar{x}_i^\top w)^2 \\ &= \frac{1}{n} \sum_{i=1}^n w^\top \bar{x}_i \bar{x}_i^\top w \\ &= w^\top \left[\frac{1}{n} \sum_{i=1}^n \bar{x}_i \bar{x}_i^\top \right] w \\ &= w^\top \left[\frac{1}{n} \bar{X} \bar{X}^\top \right] w \\ &= w^\top C w \end{aligned}$$

dove

$$t_i = (x_i - c)^\top w \quad \bar{X} = [\bar{x}_1, \dots, \bar{x}_n] \quad \bar{x}_i = (x_i - \bar{x})$$

e in particolare, C è della *matrice della covarianza*

Dunque abbiamo ottenuto che la varianza è data da:

$$\mathbf{Var}[t] = w^\top C w$$

dove C è la matrice della covarianza, definita come:

$$C = \begin{bmatrix} x_1 x_1 & \dots & x_1 x_n \\ x_2 x_1 & & x_2 x_n \\ \vdots & \ddots & \vdots \\ x_n x_1 & \dots & x_n x_n \end{bmatrix}$$

2.0.2 Autovalori e autovettori

Teorema 1: Condizioni soluzione a sistema

Dato un sistema lineare nella forma $Ax = 0$, allora questo presenta soluzioni diverse da 0 se e solo se

$$\det(A) = 0$$

Definizione 3: *Autovettore e autovalore*

Data una matrice A quadrata di dimensione $n \times n$ allora x si dice un suo autovettore relativo al valore λ se vale che

$$Ax = \lambda x$$

In particolare, per trovare autovettori e autovalori posso risolvere la seguente:

$$\det(A - \lambda I) = 0$$

questo perchè:

$$Av = \lambda v$$

$$Av - \lambda v = 0$$

$$(A - \lambda I)v = 0$$

la quale ha soluzioni non nulle se e solo se $\det(A - \lambda I) = 0$

Teorema 2: *Numero autovettori*

Data una matrice quadrata di dimensione $n \times n$, allora questa ha al massimo n autovettori linearmente indipendenti.

Teorema 3: *Numero autovettori in matrice simmetrica*

Data una matrice simmetrica quadrata di dimensione $n \times n$, allora questa ha esattamente n autovettori linearmente indipendenti.

2.0.2 Decomposizione di matrici

Data una matrice $A \in \mathbb{R}^{m \times m}$ simmetrica

- Allora esiste $U = [\mathbf{u}_1, \dots, \mathbf{u}_m] \in \mathbb{R}^{m \times m}$ and $\boldsymbol{\Lambda} = (\lambda_1, \dots, \lambda_m)^\top \in \mathbb{R}^m$ tale per cui:

$$A = U\Lambda U^\top$$

- $U \in \mathbb{R}^{n \times n}$ una matrice che ha per colonne gli *autovettori*
- $\Lambda \in \mathbb{R}^{n \times n}$ una matrice che ha sulla diagonale *gli autovalori* ordinati
- $U^\top U = UU^\top = I$

- O in alternativa

$$A = \sum_{j=1}^m \lambda_j \mathbf{u}_j \mathbf{u}_j^\top$$

- u_j autovettore
- λ_j il corrispondente autovalore

- Nota che gli *autovalori* vengono presi in ordine $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_m$

2.0.3 Obiettivo

L'obiettivo del nostro problema è trovare un sistema di riferimento in cui il primo asse sia parallelo alla direzione con varianza massima, formalmente:

$$w_i = \operatorname{argmax} \{w^\top C w : w^\top w = 1, \quad w \perp w_j \quad \forall j : 1 \leq j \leq i-1\}$$

si può dimostrare che la soluzione a questo problema di massimizzazione è dato dagli *autovettori* di C . In particolare, la i -esima componente principale è data dall' i -esimo autovettore di C . Per trovare gli autovettori di C ho due metodi:

Eigenvalue decomposition : data $A \in \mathbb{R}^{m \times m}$ *simmetrica* allora esiste

- $U = [u_1, \dots, u_m] \in \mathbb{R}^{m \times m}$
- $\Lambda = [\lambda_1, \dots, \lambda_m]$

tale per cui

$$A = U \Lambda U^\top$$

dove

$$\Lambda = \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_m \end{bmatrix}$$

SVN (Singular Value Decomposition) : data $A \in \mathbb{R}^{m \times n}$, allora esistono:

- $U = [u_1, \dots, u_m] \in \mathbb{R}^{m \times k}$
- $s = [s_1, \dots, s_n] \in \mathbb{R}^k$

tali per cui

$$A = U S V^\top$$

dove

$$\Lambda = \begin{bmatrix} s_1 & & \\ & \ddots & \\ & & s_m \end{bmatrix}$$

2.0.4 Equivalenza metodi:

Dato il fatto che $C = \frac{1}{n} \bar{X} \bar{X}^\top$, allora posso

- Chiamare $U, \lambda = \operatorname{eig}(C)$
- Chiamare $U, s, V = \operatorname{SVN}(\bar{X})$ e calcolare

$$\lambda = \left[\frac{\lambda_1^2}{n}, \dots, \frac{\lambda_n^2}{n} \right]$$

L'equivalenza dei due metodi è data dal fatto che:

$$C = \frac{1}{n} \bar{X} \bar{X}^\top = \frac{1}{n} U S V (U S V)^\top = \frac{1}{n} (U S V) (V^\top S^\top U^\top) = \frac{1}{n} U S^2 U^\top$$

quindi

$$C = \frac{1}{n} U S^2 U^\top$$

Algoritmo 2: *PCA with eigenvalue decomposition***Input:** Data points $X = [x_1, \dots, x_n]$ **Output:** Principal components $U = [u_1, \dots, u_m], \lambda = [\lambda_1, \dots, \lambda_m]$ **pca_eigen(X):**

▷ Centring points

$$\bar{X} = X - \frac{1}{n}X1_n1_n^\top;$$

▷ Compute covariance matrix

$$C = \frac{1}{n}\bar{X}\bar{X}^\top;$$

▷ Eigenvalue decomposition

$$U, \lambda = \text{eig}(C);$$

return U, λ ;dove 1_n è un vettore *colonna* di dimensione n con tutti gli elementi uguali a 1.**Algoritmo 3:** *PCA with SVD***Input:** Data points $X = [x_1, \dots, x_n]$ **Output:** Principal components $U = [u_1, \dots, u_k], \lambda = [\lambda_1, \dots, \lambda_n]$ **pca_svd(X):**

▷ Centring points

$$\bar{X} = X - \frac{1}{n}X1_n1_n^\top;$$

▷ SVN decomposition

$$U, s, V = \text{svd}(\bar{X});$$

▷ Computing λ

$$\lambda = \left(\frac{s_1^2}{n}, \dots, \frac{s_k^2}{n} \right);$$

return U, λ ;dove 1_n è un vettore *colonna* di dimensione n con tutti gli elementi uguali a 1.

Nota che per ridurre le dimensioni dei valori di un qualsiasi campione è sufficiente trovare la sua proiezione sulle componenti principali. Ricordando che la proiezione di b su a è data da $a \cdot b = a^\top b$, possiamo generalizzare al nostro caso.

Dato $\hat{W} = [w_1, \dots, w_k]$ contenente le componenti principali e $\bar{X} [x_1, \dots, x_n]$ contenente i campioni da proiettare, allora

$$T = \hat{W}^\top \bar{X}$$

T ha come colonne le proiezioni dei campioni sulle componenti principali

2.0.6 Trovare di quanto ridurre

Al fine di trovare un numero di dimensioni opportuno per cui si vuole effettuare la riduzione è possibile stimare la quantità di informazione persa con le seguenti formule:

$$\frac{\sum_{j=1}^k \lambda_j}{\sum_{j=1}^m C_{jj}}$$

Eigenvalue decomposition

$$\frac{\sum_{j=1}^k s_j^2}{\sum_{ij} \bar{X}_{ji}^2}$$

Singular value decomposition

queste due formule stimano la percentuale di informazione che viene ritenuta dopo aver ridotto la dimensione a k .

3 Clustering

3.1 k-means

- Dati dei punti $X = [x_1, \dots, x_n] \in \mathbb{R}^{d \times n}$
- Impostato un numero di *cluster* k
- Partiziona i punti in k cluster minimizzando la *variazione* totale

$$\min_{\mathcal{C}_1, \dots, \mathcal{C}_k} \sum_{j=1}^k V(\mathcal{C}_j)$$

- La variazione all'interno di un cluster \mathcal{C}_j è definita come

$$V(\mathcal{C}_j) = \sum_{i \in \mathcal{C}_j} \|x_i - \mu_j\|^2$$

- Il centroide è solitamente dato dal "centro di massa" fra i punti:

$$\mu_j = \frac{1}{|\mathcal{C}_j|} \sum_{i \in \mathcal{C}_j} x_i$$

3.1.1 Pseudocodice

Algoritmo 4: *k-means*

```
k_clustering(X):  
  ▷ Initialization strategy to get initial cluster centroids  
     $\mu_1, \dots, \mu_k$   
  while clusters change do  
    for  $x_i \in X$  do  
      Assign  $x_i$  to the closest centroid, i.e.  
       $\mathcal{C}_j = \{i \in \{1, \dots, n\} : j = \arg \min_{\ell} \|x_i - \mu_{\ell}\|\}$   
      Compute new cluster centroids  $\mu_1, \dots, \mu_k$ ;  
  return  $\mathcal{C}_1, \dots, \mathcal{C}_k$ ;
```

3.1.2 Note

- Algoritmo termina sempre, ma potrebbe finire in *minimo locale*, non globale. La *loss function* non è convessa
- Risultati variano molto in base a come si scelgono i cluster iniziali. I metodi più comuni sono:
 - Random points (not examples) in the space
 - Randomly pick examples
 - Points least similar to any existing center (furthest centers heuristic)
 - Try out multiple starting points
 - Initialize with the results of another clustering method
- Complessità ad ogni iterazione:
 - $O(kn)$ per assegnare ognuno degli n punti ad un cluster k
 - $O(n)$ computazione centroidi
- Talvolta la distanza euclidea è poco adeguata, vedi [sezione 5.1](#)
- L'assunzione di base sulla distribuzione dei dati è che questi siano disposti in maniera "sferica" (generalizzabile a più dimensioni)

3.2 EM-clustering

L'EM assume che i dati provengano da una **miscela di distribuzioni gaussiane** (invece di cluster sferici come K-means). Ogni cluster è rappresentato da una gaussiana che può avere forma ellittica, permettendo di catturare cluster con forme più complesse.

- A differenza del K-means (assegnazione *hard*), qui un punto può appartenere a più cluster con probabilità diverse
- Ad esempio un punto può avere probabilità $p(\text{rosso}) = 0.8$ e $p(\text{blu}) = 0.2$

L'algoritmo alterna tra due passi principali:

- Per ogni punto x_i , calcola la *probabilità* $p(\theta_c|x_i)$ che appartenga a ciascun cluster c
- Ricalcola i parametri θ_c di ogni cluster (centro μ_c e matrice di covarianza Σ_c) usando le probabilità calcolate nel passo precedente
- Ogni punto contribuisce al nuovo centro in modo *pesato* secondo la sua probabilità di appartenenza

L'algoritmo continua ad alternare i due passi fino a convergenza:

$$\text{E-step: } p(\theta_c|x_i) = \frac{p(x_i|\theta_c)p(\theta_c)}{\sum_{j=1}^K p(x_i|\theta_j)p(\theta_j)} \quad (1)$$

$$\text{M-step: } \theta_c^{(\text{new})} = \arg \max_{\theta_c} \sum_{i=1}^N p(\theta_c|x_i) \log p(x_i|\theta_c) \quad (2)$$

L'algoritmo garantisce che la likelihood dei dati aumenti ad ogni iterazione (anche se può convergere a un ottimo locale).

Nota: $p(x_i | \theta_c)$ è la funzione gaussiana valutata in x_i . La generalizzazione della gaussiana in d dimensioni è:

$$\mathcal{N}[x; \mu, \Sigma] = \frac{1}{(2\pi)^{d/2} \sqrt{\det(\Sigma)}} \exp \left[-\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu) \right]$$

dove Σ è la *matrice di covarianza* che descrive la forma dell'ellisse del cluster e μ è la *media*

3.3 Spectral clustering

Lo spectral clustering può aiutare nel momento in cui i dati non sono distribuiti in maniera gaussiana o circolare. L'algoritmo funziona così:

- Costruzione grafo secondo qualche criterio
- Partizione del grafo minimizzando il peso del taglio, ossia

$$\text{cut}(A, B) = \sum_{x \in A, y \in B} w(x, y)$$

più nello specifico, si cerca di minimizzare il taglio normalizzato:

$$\text{Ncut}(A, B) = \frac{\text{cut}(A, B)}{\text{vol}(A)} + \frac{\text{cut}(A, B)}{\text{vol}(B)}$$

nota che il taglio va *normalizzato* per evitare che il taglio sia influenzato dalla dimensione dei cluster, evitando tagli in cui ho 99 nodi da una parte e 1 dall'altra

Il problema può essere generalizzato ad un problema di autovalori

3.4 Clustering gerarchico

L'idea del clustering gerarchico è di procedere in maniera *agglomerativa* e *bottom-up*, aggregando gruppi simili

- Ogni elemento ξ è inizialmente un cluster
- Unisco gruppi secondo qualche criterio
- Termina quando rimane un singolo cluster

La struttura gerarchica ottenuta è detta *dendrogramma*

Ciò che caratterizza un algoritmo è proprio come si definisce la distanza fra clusters. Ci sono molte opzioni quali: closest pair, farthest pair, average among all distances, etc

3.4.1 Pseudocodice

Formalmente, per risolvere il problema descritto in eq. (2), è necessario procedere come segue:

$$\begin{aligned}N_c &= \sum_{i=0}^k p(x \mid \theta_c) \\ \Sigma_c &= \frac{1}{N_c} + \sum_{i=1}^n p(\theta_c \mid x_i) \cdot \bar{x}_i \bar{x}_i^\top \\ \pi_c &= p(\theta_c) = \frac{N_c}{n}\end{aligned}$$

Algoritmo 5: *EM-clustering*

Input: Data points $X = [x_1, \dots, x_n] \in \mathbb{R}^{d \times n}$, number of clusters K

Output: K clusters with parameters $\theta_c = (\mu_c, \Sigma_c, \pi_c)$

em_clustering(X, K):

▷ Inizializzo π

for $c \in K$ **do**

└ $\pi_i = \frac{1}{K}$

while clusters change **do**

▷ E-step: calcolo le probabilità di appartenenza

for $c \in K$ **do**

$$\gamma_{iC} = p(\theta_c, \xi) = \frac{p(x_i | \theta_c) p(\theta_c)}{\sum_{j=1}^K p(x_i | \theta_j) p(\theta_j)};$$

▷ M-step: aggiorno gaussian parameters

for $c \in K$ **do**

$$N_c = \sum_{i=0}^k p(x | \theta_c);$$

$$\Sigma_c = \frac{1}{N_c} + \sum_{i=1}^n p(\theta_c | x_i) \cdot \bar{x}_i \bar{x}_i^\top;$$

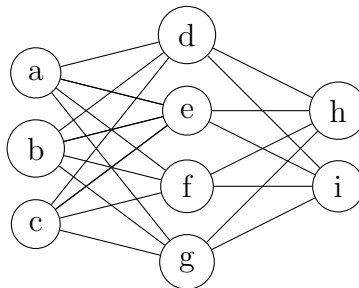
$$\pi_c = p(\theta_c) = \frac{N_c}{n};$$

3.4.2 Vantaggi rispetto a k-means

1. *Cluster ellittici*: Può gestire cluster con forme allungate, non solo circolari
2. *Clustering soft*: Fornisce probabilità di appartenenza invece di assegnazioni rigide
3. *Maggiore flessibilità*: La matrice di covarianza permette di catturare correlazioni tra le features

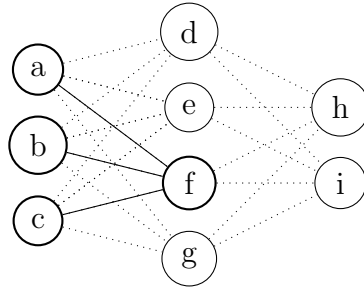
4

Reti neurali



4.1 Calcolo output predetto

Data una rete neurale con L layers, l'idea è che ogni nodo ha un'attivazione, ogni edge un peso. Per calcolare l'attivazione di un nodo t devo fare una "somma pesata" di tutti i nodi ai quali è collegato t .



in questo caso:

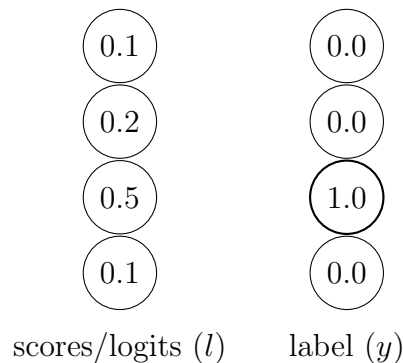
$$\text{Attivazione di } f = h(z_f = z_a + w_{af} + z_b + w_{bf} + z_c + w_{cf} + b_f)$$

dove b_f è il bias di f e h è la funzione di attivazione. In termini generali:

$$z_t = \sum_{i \in \mathcal{N}(t)} w_{it} z_i + b_t$$

4.1.1 Classificazione: softmax + cross-entropy

Una volta calcolate le attivazioni dell'ultimo layer per confrontarle con il risultato atteso (y) posso procedere come segue. Supponendo di avere



Softmax : la funzione *softmax* è definita come:

$$\text{softmax}(l) = S(l) = \frac{\exp(l_i)}{\sum_{j=1}^k \exp(l_j)}$$

intuitivamente:

- Applica esponenziale a ogni score z_i per enfatizzare gli outliers
- Normalizza il vettore l rendendolo una distribuzione di probabilità

questo equivale a creare il vettore:

$$S(l) = \left[\frac{e^{l_1}}{\sum_j e^{l_j}}, \dots, \frac{e^{l_m}}{\sum_j e^{l_j}} \right]$$

Cross-entropy loss : la *cross-entropy* è definita come:

$$\text{cross-entropy}(l, y) = \mathcal{L}(l, y) = \sum_k y_k \log(S(l)_k)$$

In termini vettoriali, dato l il vettore degli score e y ossia la label che indica il risultato atteso, allora la *cross-entropy* è definita come:

$$\mathcal{L}(l, y) = S(l)^\top y$$

Esempio : nel nostro caso:

$$S(l) = S([0.1, 0.2, 0.5, 0.1]) = \frac{1}{\sum_j e^j} [e^{0.1}, e^{0.2}, e^{0.5}, e^{0.1}]$$

calcolo la sommatoria:

$$\sum_i e^{l_i} = e^{0.1} + e^{0.2} + e^{0.5} + e^{0.1} \approx 3.2$$

dunque

$$S(l) = \frac{1}{3.2} [1.1, 1.2, 1.6, 1.1] \approx [0.34, 0.38, 0.5, 0.34]$$

4.1.2 Regressione: softmax + cross-entropy

Se il vettore l degli scores deve dare origine ad una regressione di n valori allora posso calcolare la distanza dal valore atteso \hat{y} con la loss quadratica:

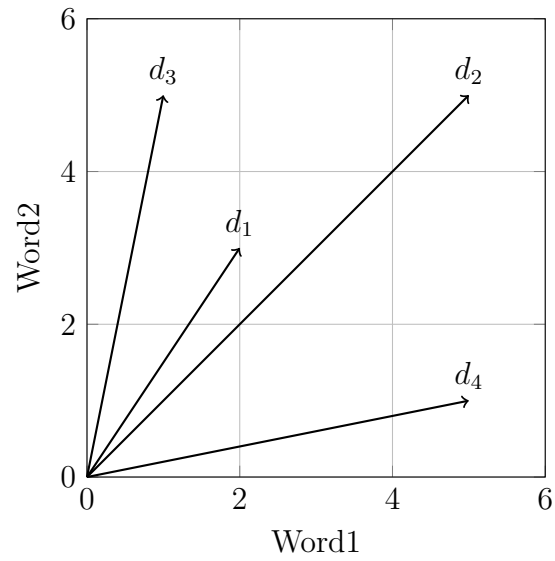
$$\mathcal{L}(l, \hat{y}) = \frac{1}{2} \sum_{i=1}^n (l_i - \hat{y}_i)^2$$

nota che questa loss function è relativa *al singolo input* x_i

5 Miscellanea nozioni

5.1 Cosine distance

In alcuni casi la distanza euclidea non è adeguata per valutare quanto due vettori sono simili. Ad esempio, se classificassimo documenti sulla base delle parole contenute e avessimo:



in questo caso d_1 e d_2 sono molto distanti mentre dal punto di vista logico risultano molto simili. Per questo è più efficiente la *cosine similarity*

$$\text{sim}(x, y) = \frac{x}{||x||} \cdot \frac{y}{||y||}$$

nota come $\text{sim}(x, y) \in [0, 1]$. Similmente:

$$d(x, y) = 1 - \text{sim}(x, y)$$