Document Semantic Similarity

TIS Project

Alberto Pirovano Francesco Picciotti

Politecnico di Milano

21st May 2017

Outline

Introduzione

Le tecniche adottate attualmente per trovare la **similitudine semantica tra testi** si basano su tre approcci:

1. NLP Tradizionale
2. Vector Space Model
3. Deep Learning based

Outline

NLP Tradizionale

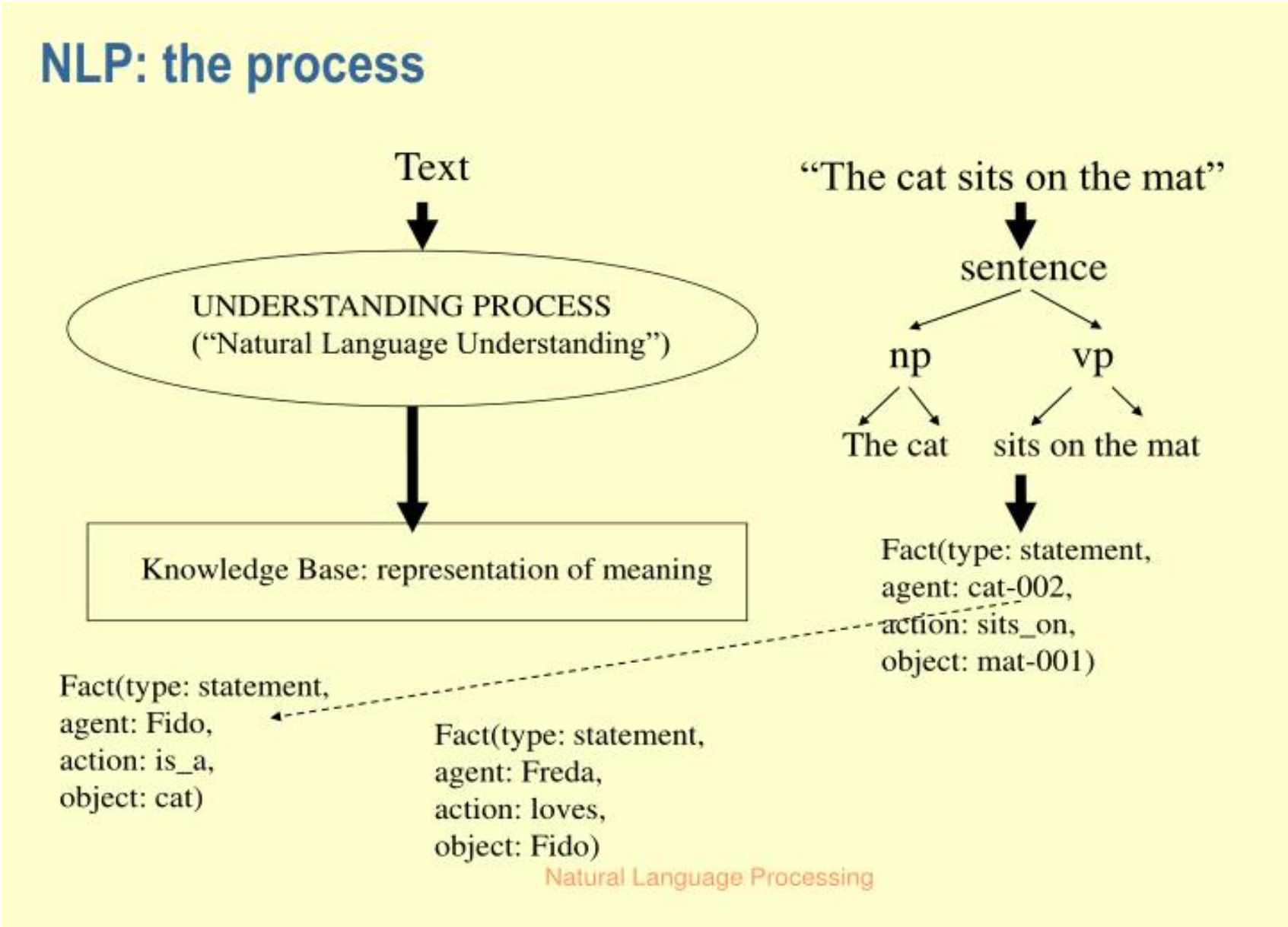
Questo approccio si basa sull'utilizzo delle tradizionali tecniche di **Natural Language Processing** e si costituisce dei seguenti step:

- Cleaning dei dati
- Pos-Tagging
- Stemming o Lemmatisation
- Parsing
- Ontologia

Tuttavia, dato che il nostro lavoro è molto **sensibile** e **dipendente** dalla qualità dei tool utilizzati, abbiamo trovato alcune consistenti **criticità** riguardanti:

- **L'affidabilità** del Pos-Tagger italiano di TreeTagger
- **Reperire** una Ontologia e un parsing tool per la lingua italiana

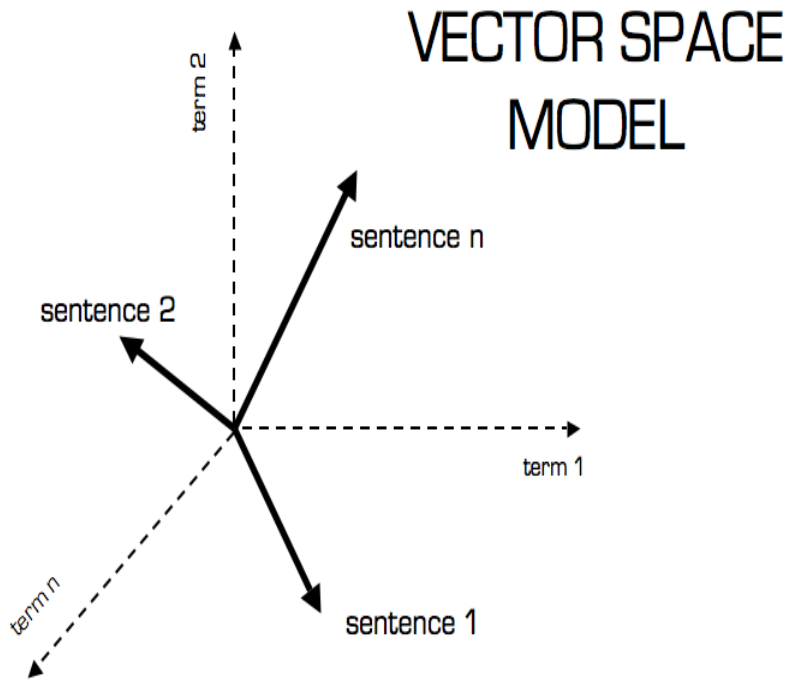
NLP Tradizionale



Outline

Vector Space Model - Explanation

Differentemente dal precedente, questo approccio ha le sue basi nello sviluppo di una **rappresentazione geometrica e vettoriale** per i documenti testuali. **Documenti e query** sono rappresentati da vettori con un numero di elementi pari al numero di termini presenti nel vocabolario. Tipicamente i termini sono le **parole distinte** presenti nell'insieme di documenti. A valle di questa rappresentazione vengono spesso utilizzate le **operazioni vettoriali** per confrontare due **documenti**.



Vector Space Model - Encoding

Nel vector space model proposto da **Salton, Wong and Yang** i vettori sono composti da **weights**, ognuna associata ad un termine del dizionario e calcolata tramite **tf-idf**.
Considerando un **documento** d_j, questo viene rappresentato tramite un **vettore** d_j:

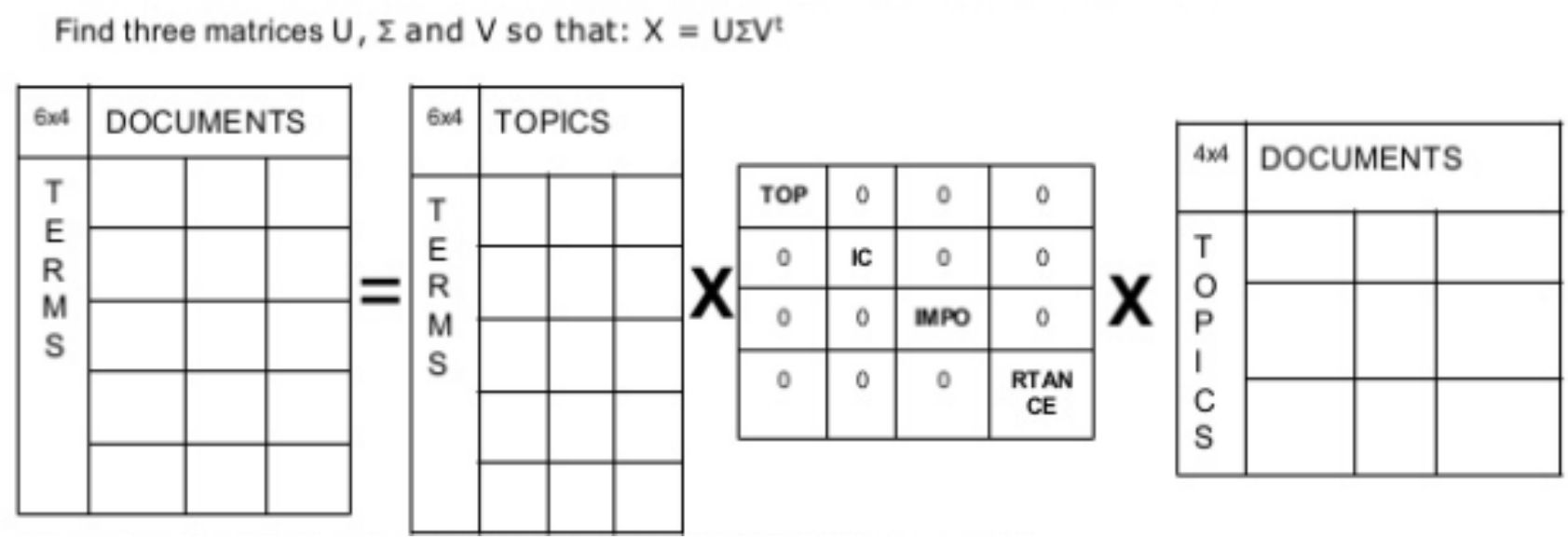
- $d_j = (w_{1,j}, w_{2,j}, ..., w_{t,j})$ **dove**: $w_{i,j} = tf_{t,d} * \log \frac{|D|}{|\{d' \in D | t \in d'\}|}$

Vector Space Model - LSA

Latent Semantic Analysis, anche detta LSI in information retrieval, è una tecnica di **Topic Modelling** che si colloca a valle del **document encoding** con **tfidf**.
È una tecnica di **feature extraction** che permette di migliorare significativamente la qualità di un lavoro di **clustering**.
Questa procedura viene usata per generare una categorizzazione di un set di documenti in un set di topic o anche per osservare le parole che descrivono un certo topic.
Si basa sulla creazione di una **Document-Term Matrix** nella quale le **righe** rappresentano le parole del **Bag Of Words** e ha una **colonna** per **documento** nel corpora.
Il cuore di questa procedura sta nella **riduzione della dimensionalità** di questa matrice tramite **SVD**.

Vector Space Model - LSA - SVD

- **U:** ogni riga rappresenta una parola, mentre ogni colonna rappresenta una **dimensione nel latent space**; le colonne sono **ortogonali** l'una all'altra e sono **ordinate** in base alla varianza dei dati lungo di esse.



È importante sottolineare che usando solo **k** delle **m** dimensioni latenti si ottiene una **approssimazione** di **X**.

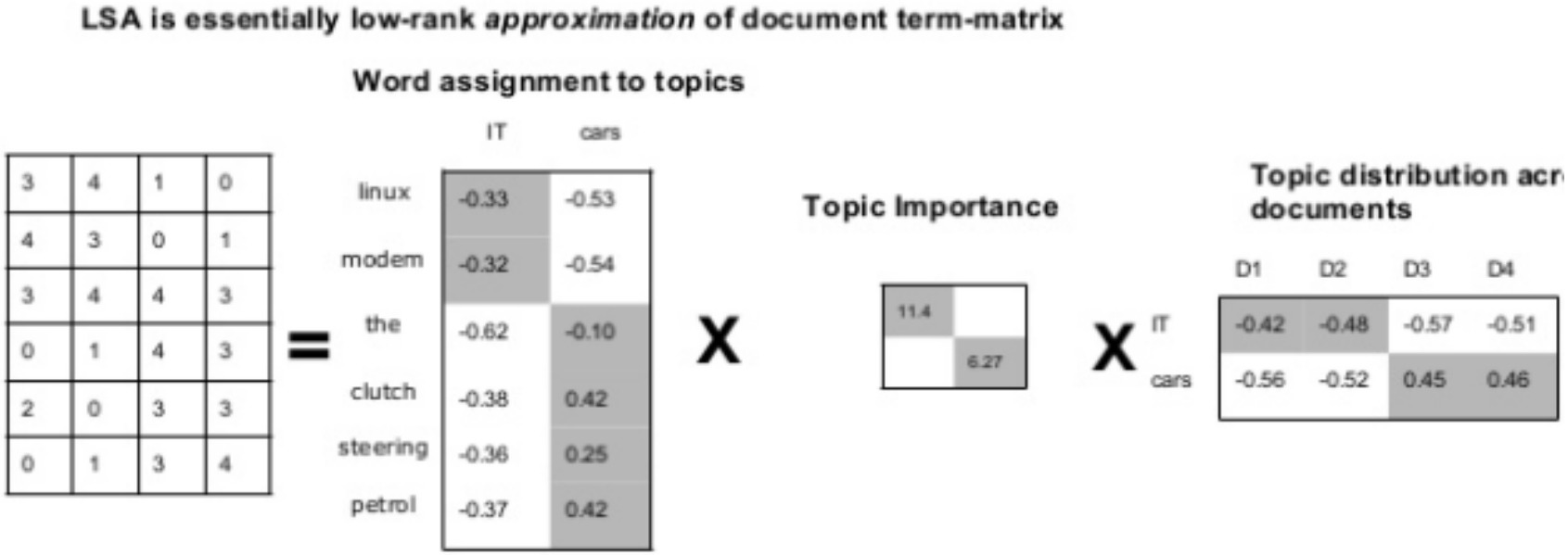
Vector Space Model - LSA

Questa procedura ci permette di:

- **Estrarre** quanti topic desideriamo da un set di documenti.
- **Conoscere** la rilevanza di un certo topic dopo averlo estratto, in questo modo siamo in grado di fermare il processo di estrazione quando i topic cominciano a diventare poco significativi.
- **Categorizzare** documenti in topic
- **Descrivere** topics con le parole del **Bag Of Words**.

Vector Space Model - LSA - Example

In questo esempio possiamo osservare la riduzione di dimensionalità:



Il processo di LSA permette di costruire le 3 matrici che vediamo sopra, ognuna con una sua utilità:

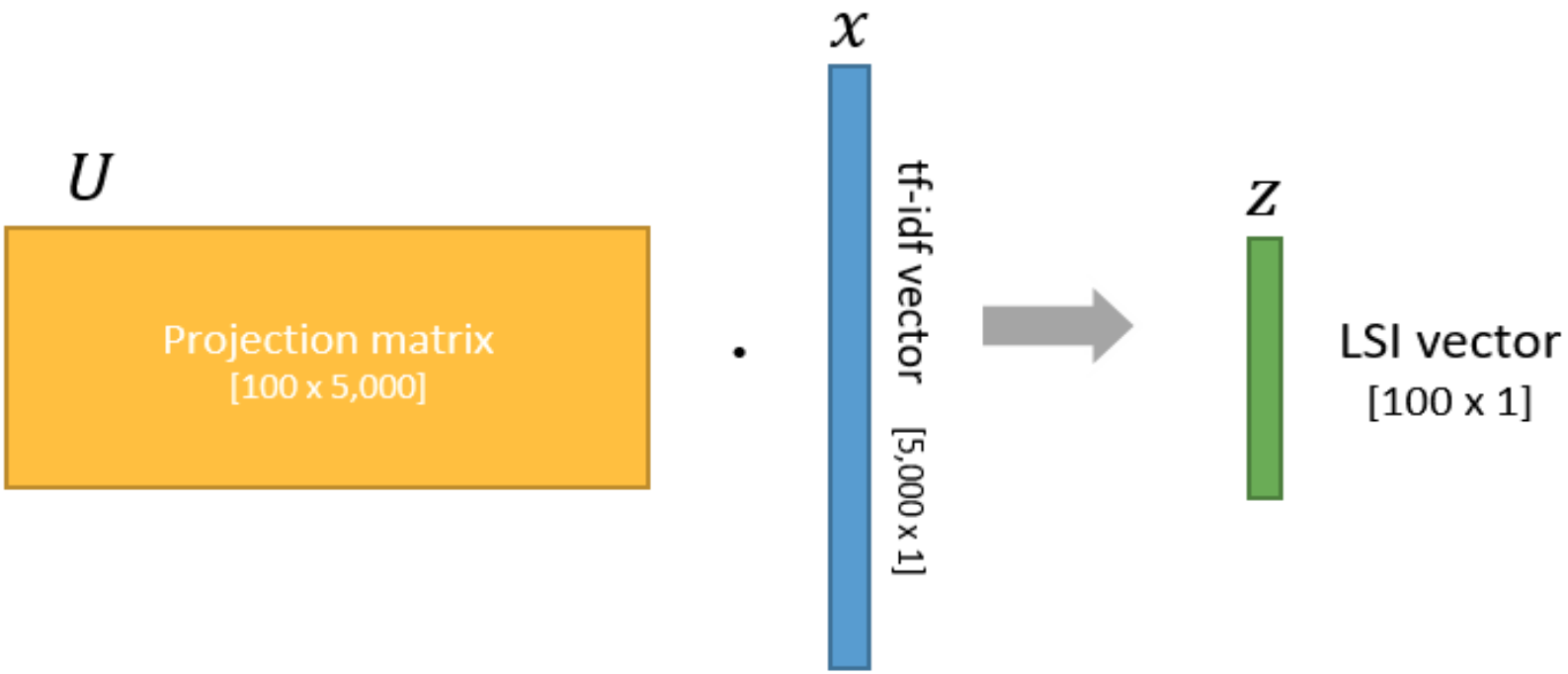
1. **Word assignment to topics**
2. **Topic importance**
3. **Topic distribution across documents**

Vector Space Model - LSA - Example

Ipotizzando un numero di topics pari a 100 e la cardinalità del **Bag of Words** pari a 5000, per ottenere la rappresentazione dei documenti in termini di topics, cioè un vettore **z**, dobbiamo:

1. **Vettorizzare** un documento in **x**.
2. **Proiettare** il **tfidf** vector **x** sul topic space.

Se definiamo la matrice $U = (Wordassignmenttotopics)^T$ possiamo visualizzare la **proiezione** in questo modo:



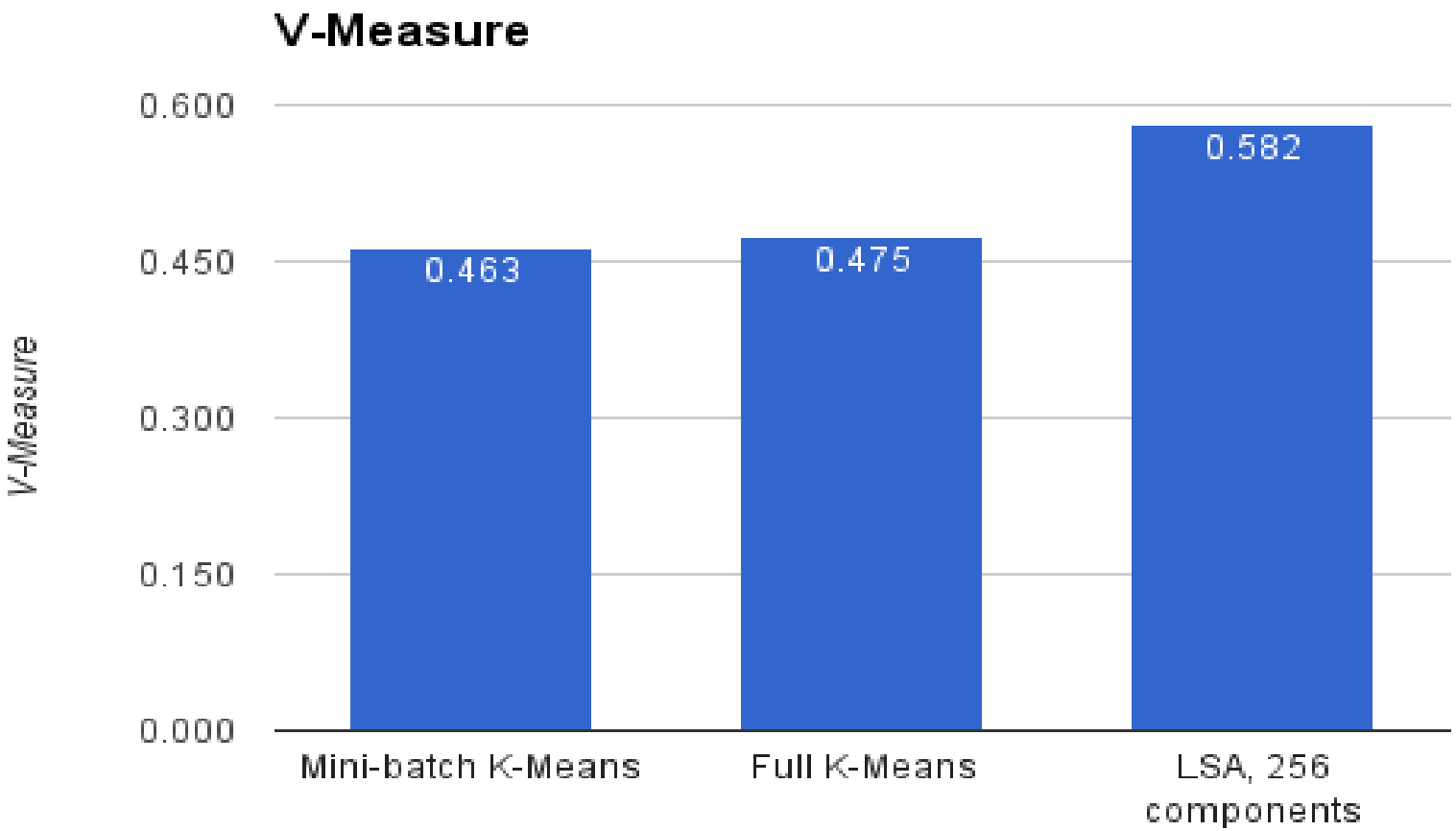
Vector Space Model - Algorithm

Riassumendo, possiamo individuare i seguenti **passaggi**:

1. Cleaning dei dati (Stemming o Lemmatisation)
2. Vectorization using TF/IDF (**x** nella figura nella slide precedente)
3. LSA (optional) (**z** nella figura precedente)
4. Clustering using similarity measure (Cosine, Pearson, ...) tra **x** vectors o **z** vectors

Vector Space Model - LSA - Performance

Come possiamo vedere nella figura, utilizzare le classiche tecniche di **clustering** senza **LSA** può portare una riduzione rilevante delle **performances**:



Vector Space Model (con LSA) - Pros and Cons

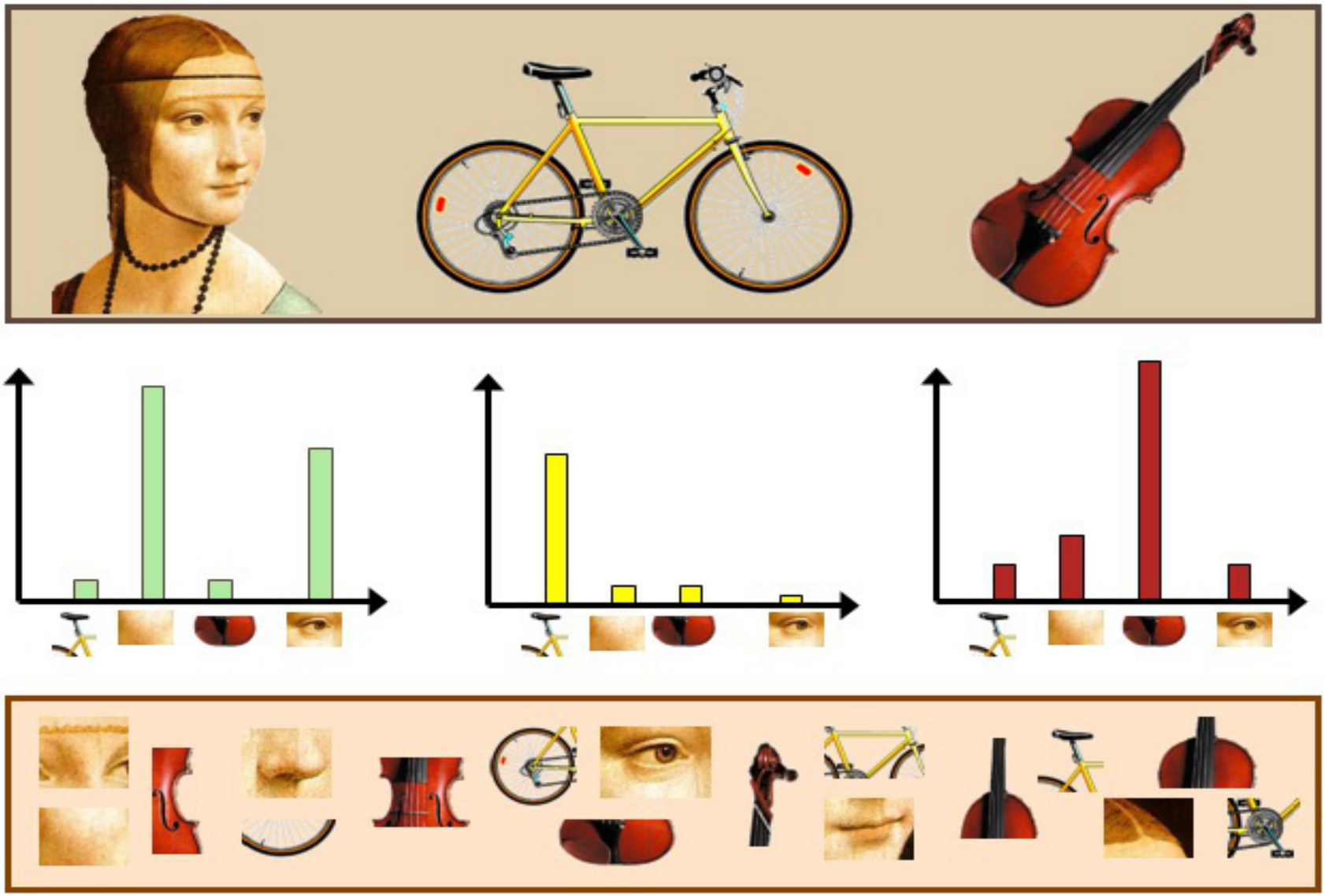
Cons:

- Non adatto a trattare **lunghi documenti**, infatti a causa della alta **dimensionalità** il valore **tfidf** dei componenti si riduce, riducendo di conseguenza il **dot-product**.(LSA fixes)
- **Fortemente sensibile a Falsi Negativi**, infatti documenti nello stesso **contesto** ma con diversa terminologia non saranno considerati simili.(LSA fixes)
- Perde **l'ordine delle parole**.
- **Pre-processing** dipendente.
- **Le dimensioni generate possono essere difficili da interpretare**, sensate matematicamente ma non dal punto di vista del natural language

Pros:

- Modello semplice basato sull'**algebra**.
- Le **weights** non sono binarie.
- Permette di calcolare un grado di **similarità continuo**.

...limiti?



Outline

Deep learning based

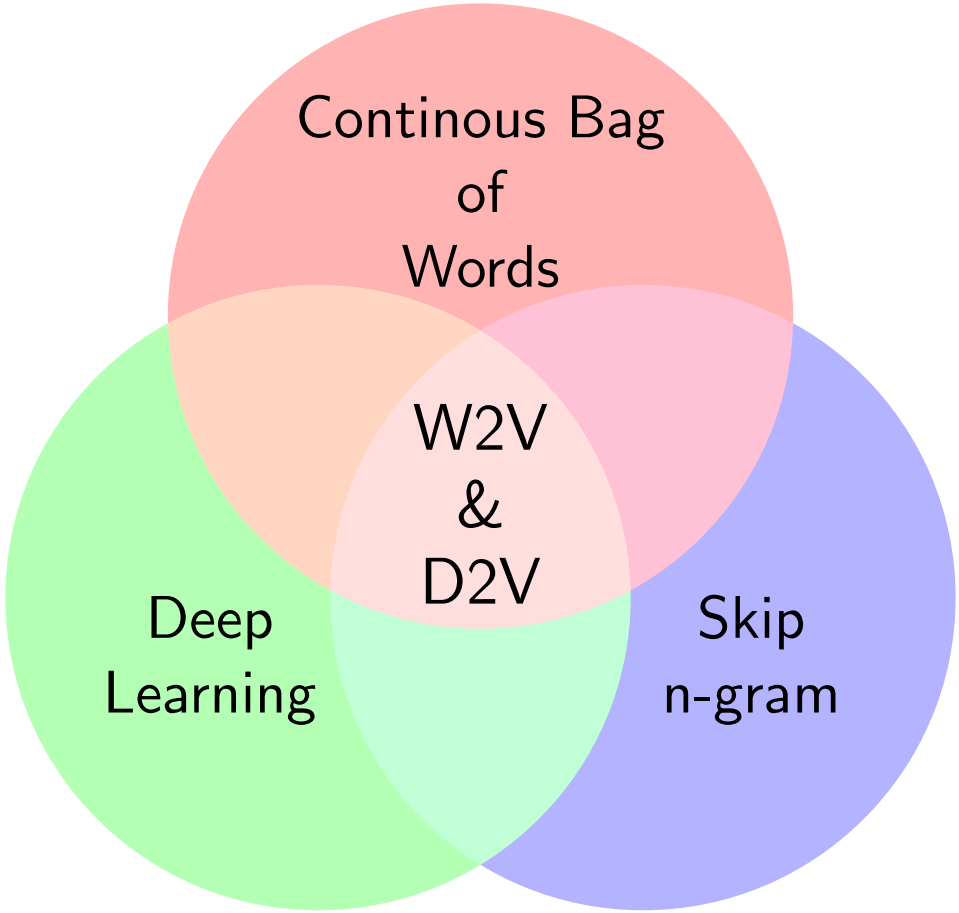
Il terzo approccio che proponiamo è molto diverso dai precedenti, infatti si basa sull'utilizzo di **neural networks**. La fondamentale differenza è che, mentre il secondo **definiva un algoritmo** per determinare le rappresentazioni vettoriali delle parole (tfidf), questo **definisce una neural network** con la task di imparare quell'algoritmo dai dati. Questa **NN** può essere **costruita** in due diversi modi, ognuno dei quali descrive **come** imparare la **word-representation** per ogni parola:

- **Continous Bag Of Words model**
- **Skip Grammar model**

Dato che il processo di apprendimento è **unsupervised**, questi modelli permettono di **determinare la task** tramite la definizione di un target per ogni input.

Deep Learning based

Questa tecnica quindi si basa su 3 ambiti:



Deep Learning based: Word2Vec & Doc2Vec

Questo metodo è stato implementato da **Google** sotto il nome di **Word2Vec** e **Doc2Vec**. Il primo permette di determinare le **relazioni semantiche** che un particolare **corpus di testi** assegna ad un **Bag Of Words** di parole. **Doc2Vec** invece è una tecnica che si configura come una **estensione di Word2Vec** la quale, preso in ingresso un set di documenti (corpora), genera un **grado di similarità**.

Deep Learning based - Pros and Cons

Dopo una ampia **discussione**, seguita da una approfondita **analisi critica** di questi due approcci, siamo giunti alle seguenti **conclusioni**, che in termini di pro e contro si possono riassumere nel seguente modo:

Pros:

- Molto meno dipente da un preprocessing
- Combina il metodo *Geometrico* con quello *NLP Tradizionale*
- **Non sfrutta una ontologia, ma la crea**
- **Language independent**
- **Tiene in considerazione anche l'ordine delle parole**, **Context-aware**

Cons:

- Tecnica **unsupervised**
- Necessità di un esperto per **validare** la similarità
- Può risultare in **GIGO** system (Garbage In Garbage Out)

Outline

Dataset

"Preprocessing is 80% of NLP work"

Lev Konstantinovskiy

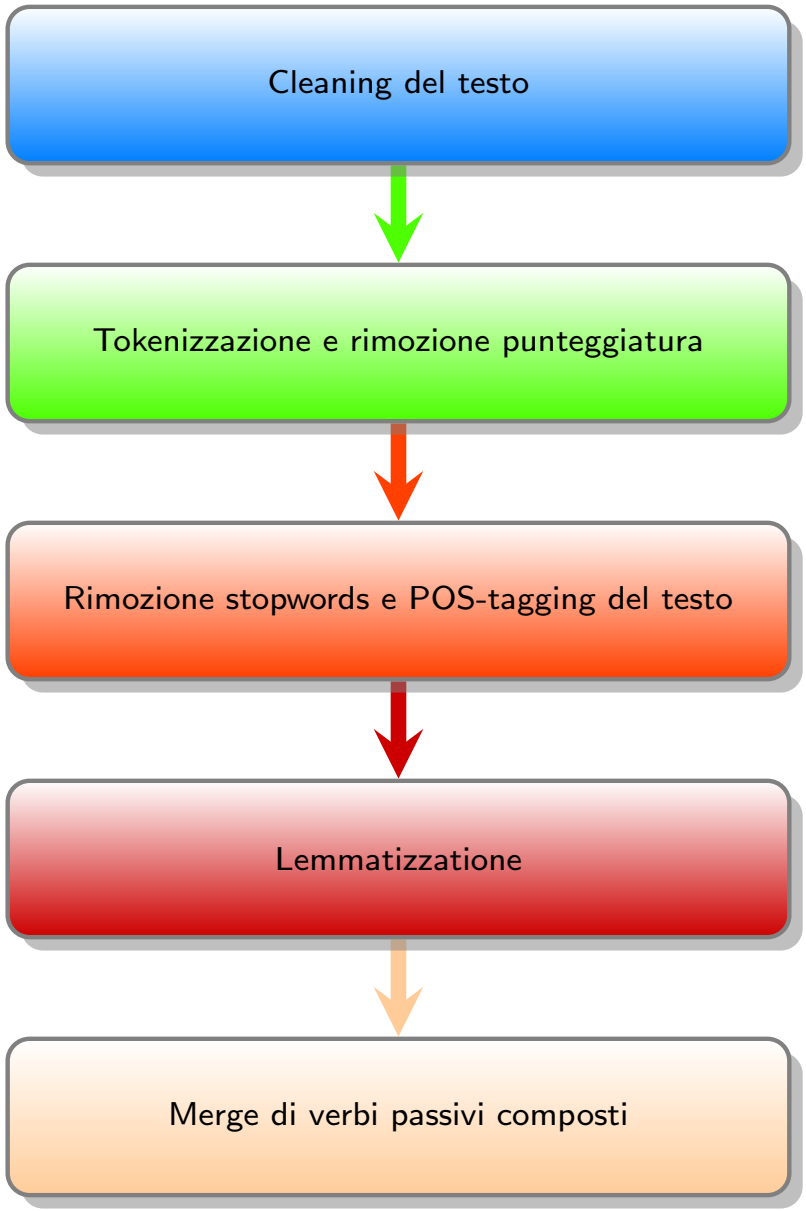
Il **dataset** si suddivide in due corpora:

- il corpus del **Sole 24 Ore** con 3265 articoli, di cui 31 non hanno body
- il corpus di **Radiocor** con 6916 articoli

Il corpus prima del **preprocessing** contiene quindi 10150 articoli.
Togliendo i **duplicati** otteniamo 9283 articoli, cioé ci sono 867 articoli duplicati.

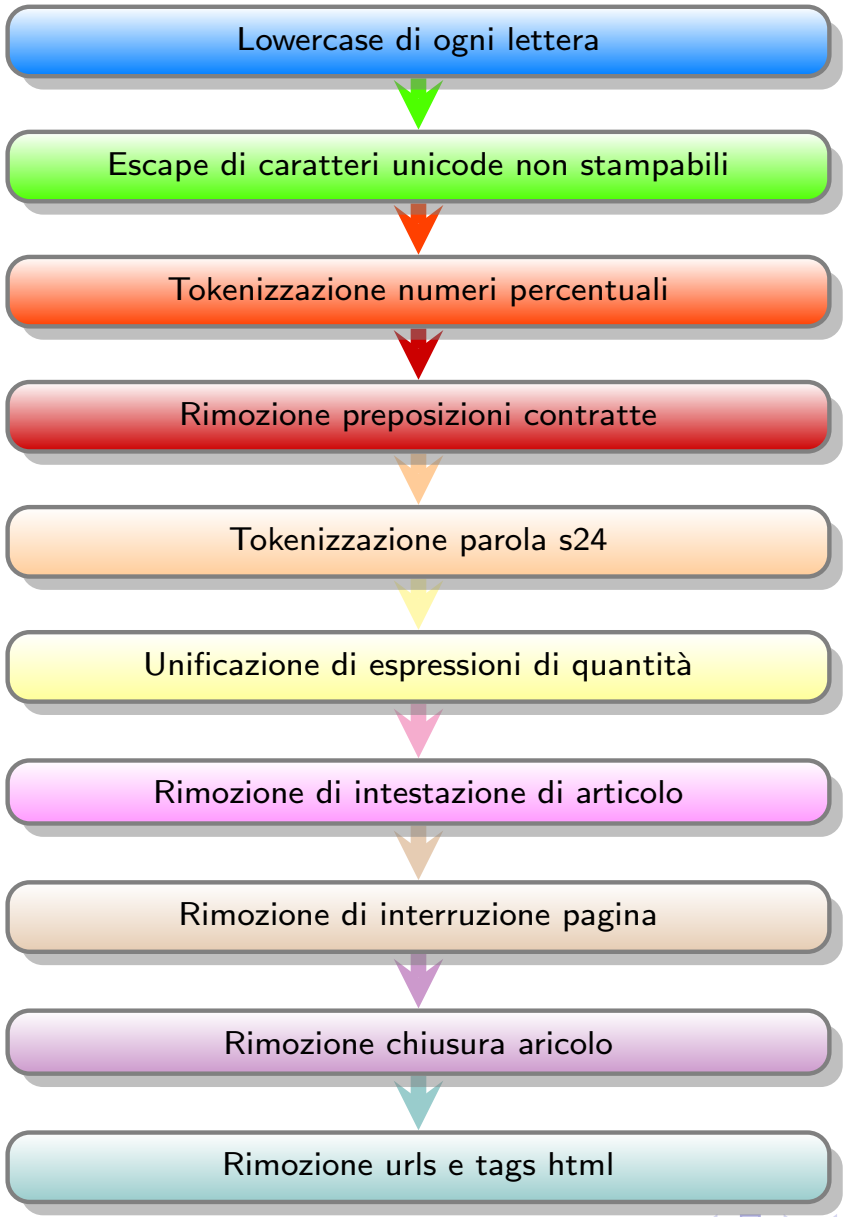
Outline

Pipeline completa



Outline

Cleaning pipeline



Outline

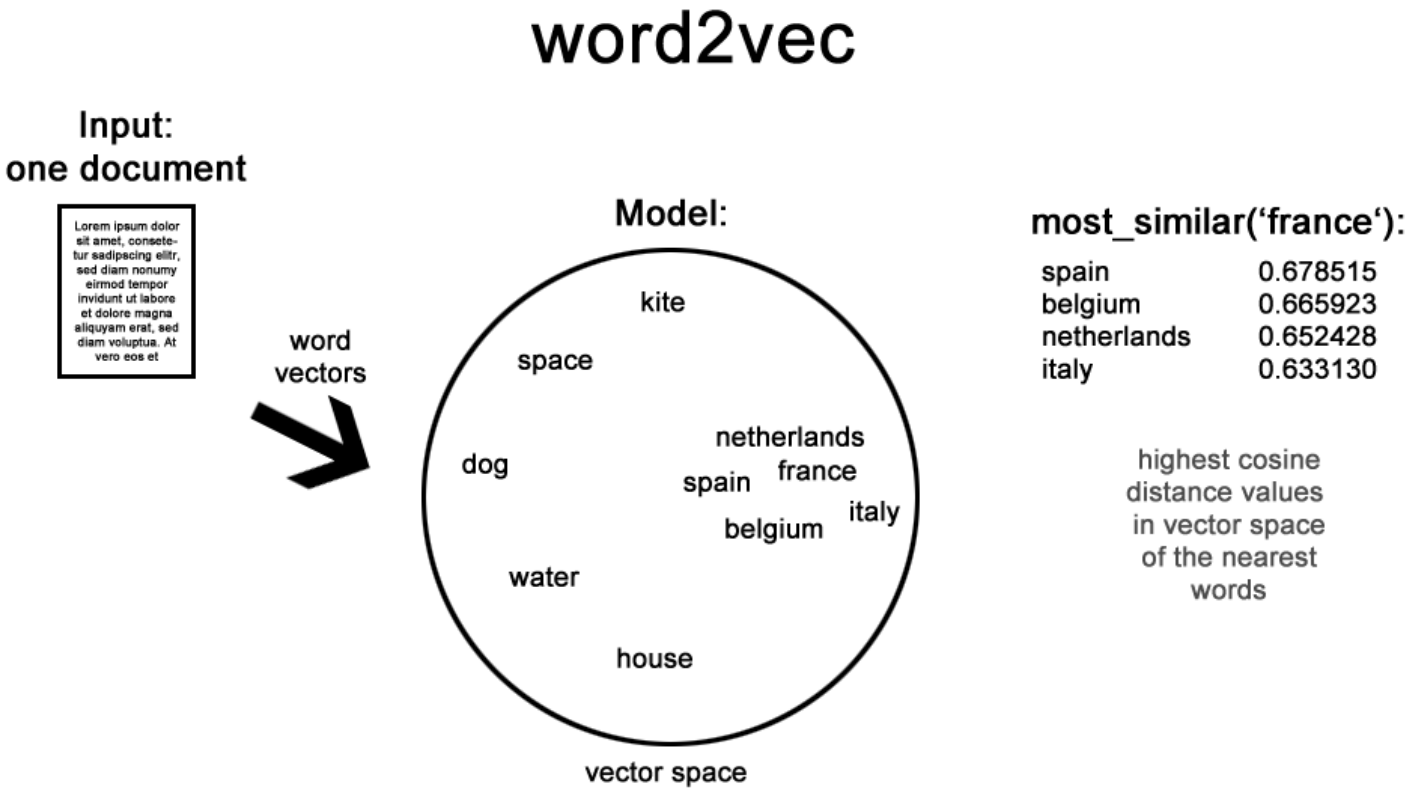
Word2Vec - Task

Word2Vec è un gruppo di modelli che vengono usati per generare **word-embeddings** da **unstructured text**. Per questo scopo viene usata una **two-layers neural network** che viene trainata per ricostruire il **contesto linguistico** delle parole.

- **Input: corpus** di testo
- **Output: vector-space**

Ogni parola **unica nel corpus** verrà rappresentata con un vettore nel **vector-space**. L'algoritmo genererà degli word-embeddings tali per cui parole che condividono lo stesso contesto nel corpus risulteranno vicine nel **vector space**.

Word2Vec - Idea



Word2Vec - From unsupervised to supervised

Questa task può essere definita in due modi:

1. **Skip Grammar** - predire il **contesto** data la parola in input
   - **Input:** focus-word
   - **Target:** context-words
2. **Continous Bag Of Words** - predire la parola in input dato il **contesto**
   - **Input:** context-words
   - **Target:** focus-word

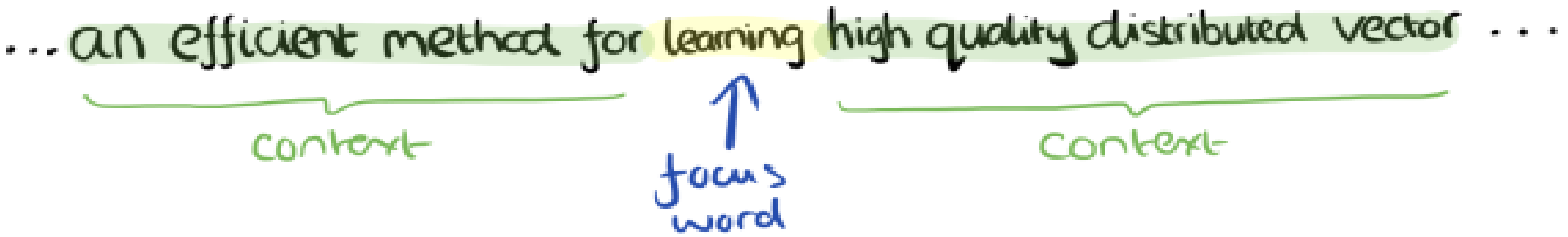
Definendo input e target pairs abbiamo definito un **supervised problem**, che affronteremo usando **logistic classifiers** (logistic regression).

Word2Vec - Encoding

Gli **input-target pairs** sono rappresentati tramite **One Hot Encoding**, mentre l'**output** della NN sarà un vettore di probabilità di dimensione pari al numero di parole del **vocabolario**.
Di conseguenza se abbiamo un vocabolario di **dimensionalità V** avremo dei **vectors** costituiti da V elementi di cui un 1 e gli altri 0.

Word2Vec - Context

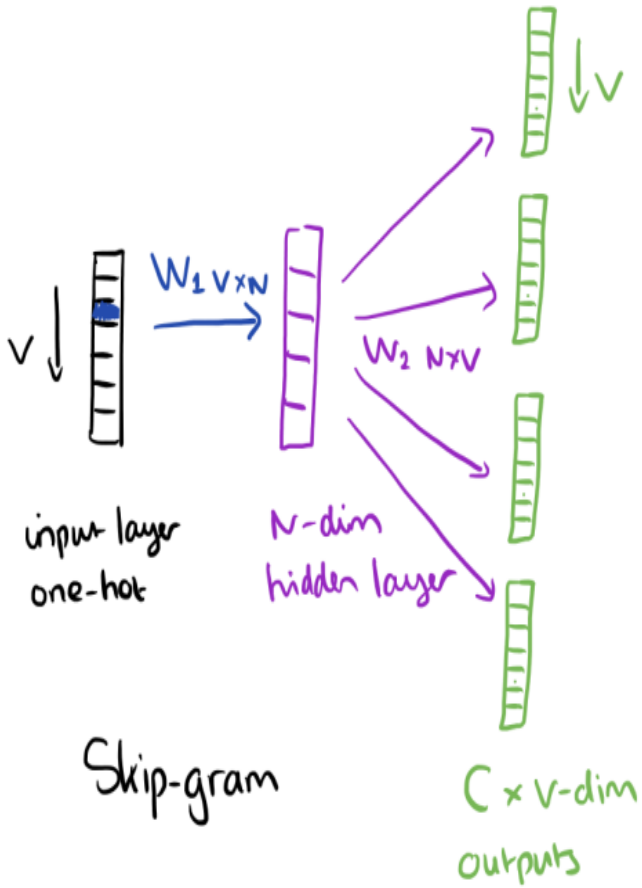
In entrambi i casi è necessario definire un **context** per la parola in input, che identifichiamo con un parametro chiamato $c = windowsize$. Tramite questa *windowsize*, una volta fissata una **input word/focus word**, chiamiamo **context** l'insieme delle c parole prima della input word più le c parole dopo la input word.



Terminologia:

- V = numero di elementi nel vocabolario
- N = numero di weights per word embeddings
- c = window size

Word2Vec - Skip Grammar



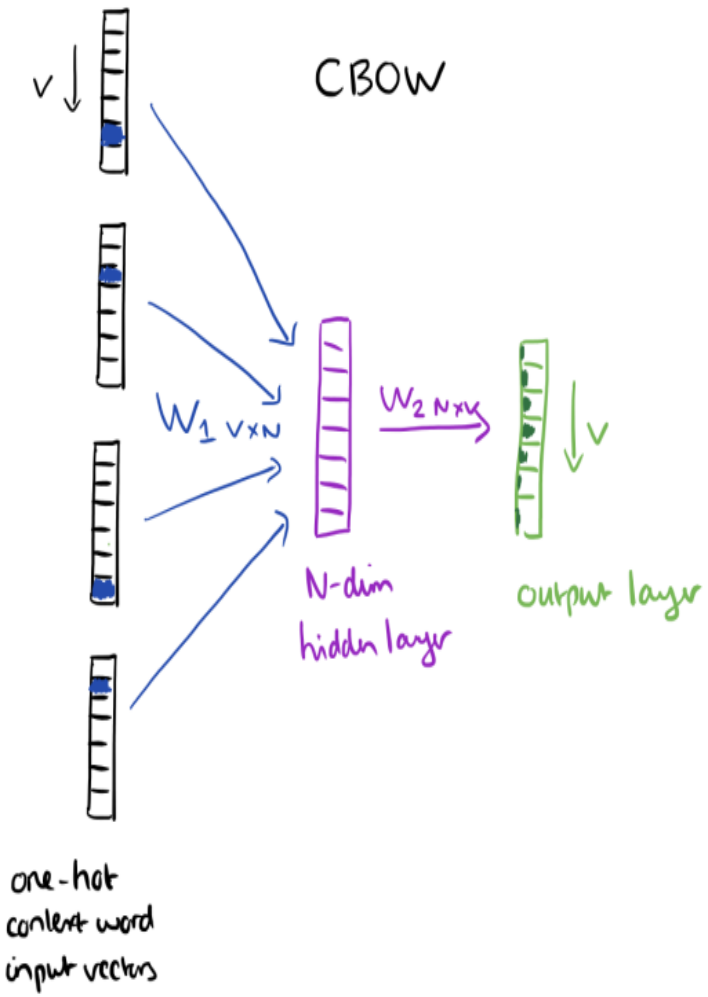
Data una sequenza di **training words** $w_1, w_2, ...w_M$, con $M = trainingsetsize$, il **training objective** di questo modello è trovare il parametro θ che massimizza la **Log-Likelihood**:

$$L = \frac{1}{M} \cdot \sum_{m=1}^{M} \sum_{-c<j<c, j\neq 0} \log p(w_{m+j}|w_m)$$

Ogni classification task, data una **focus word** w_m e una parola del contesto w_{m+j}, calcola **la probabilità** $p(w_{m+j}|w_m)$.

- Di conseguenza per ogni input vector definiamo circa **2c logistic classifiers**.

Word2Vec - CBOW



Data una sequenza di **training words** $w_1, w_2, ...w_M$, con $M = trainingsetsize$, il **training objective** di questo modello è trovare il parametro θ che massimizza la **Log-Likelihood**:

$$L = \frac{1}{M} \cdot \sum_{m=1}^{M} \log p(w_m | w_{m-c}, ..., w_{m+c})$$

Ogni classification task, date una serie di **context words**, le calcola **la probabilità** $p(w_m | w_{m-c}, ..., w_{m+c})$.

- Di conseguenza per ogni input vector definiamo **1 logistic classifier**.

Word2Vec - Matrices

Le due **weight matrices** $W_1[V \cdot N]$ e $W_2[N \cdot V]$ sono della stessa dimensione e contengono entrambe un **word-embedding vector** per ogni vocabulary word.
La **prima** fornisce la **input vector representation** delle vocabulary words e la **seconda** fornisce la **output vector representation** delle vocabulary words.
In generale data una parola $w \in V$:

- v_w è la sua **input representation**, vale a dire un vettore $[1 \cdot N]$ relativo a w che otteniamo da W_1:
- v'_w è la sua **output representation**, vale a dire un vettore $[1 \cdot N]$ relativo a w che otteniamo da W_2:

Word2Vec - Hidden layer

È interessante notare come, dato un input **OHE**, l'**hidden layer** funga da **lookup table** per selezionare il **word vector** relativo alla input word, v_{wInput}(no activation function).

$$[0 \quad 0 \quad 0 \quad 1 \quad 0] \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = [10 \quad 12 \quad 19]$$

Possiamo quindi dire che:

- **Skip Grammar:** L'**output dell'hidden layer** (v_{wInput}) è il **word vector** relativo all'**input word**, in questo caso $v_{wInput} = [10, 12, 19]$.
- **CBOW:** Ogni **OHE** vector delle parole del contesto effettua il lookup e ottiene il word-embedding dalla matrice W_1. Successivamente viene fatta una media degli word embeddings ottenuti in modo tale da ottenere l'output dell'hidden layer ($v_{wAverage}$).

Word2Vec - Probabilistic view

Considerando i due tipi di approccio, possiamo quindi esplicitare le **probabilità scalari** utilizzate nella funzione di costo, con $w_m = w_{Input}$:

- **Skip Grammar:** $p(w_{m+j}|w_m) = \frac{exp(v'^T_{w_{m+j}} \cdot v^T_{w_{Input}})}{\sum_{i=1}^{V} exp(v'^T_{w_i} \cdot v^T_{w_{Input}})}$
- **CBOW:** $p(w_m|w_{m-c}, ..., w_{m+c}) = \frac{exp(v'^T_{w_m} \cdot v^T_{wAverage})}{\sum_{i=1}^{V} exp(v'^T_{w_i} \cdot v^T_{wAverage})}$

Tecnicamente il **vettore di probabilità** che viene generato dalla rete, dato un **input vector** o un **input average**, è:

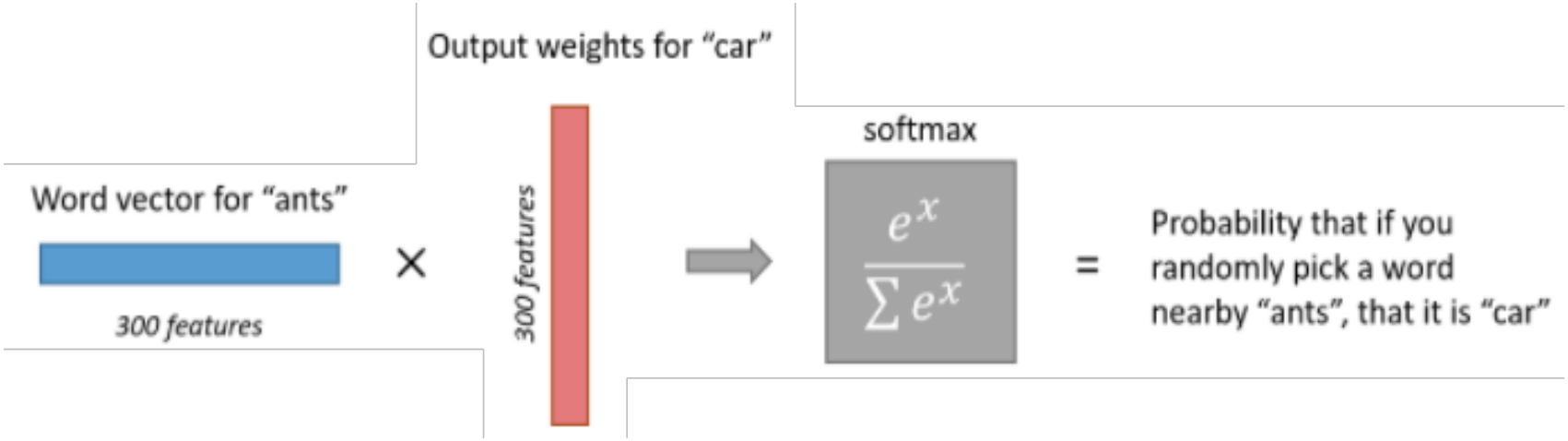
- **Skip Grammar:** $y_{t+j} = \frac{exp(W_2^T \cdot v^T_{wInput})}{\sum_{i=1}^{V} exp(v'^T_{w_i} \cdot v^T_{wInput})}$
- **CBOW:** $y_t = \frac{exp(W_2^T \cdot v^T_{wAverage})}{\sum_{i=1}^{V} exp(v'^T_{w_i} \cdot v^T_{wAverage})}$

Word2Vec - Output layer

L'**output layer** si compone di due operazioni:

1. Calcolare lo score dell'input vector, $scoreVec = v_{wInput}^T \cdot W_2$ per **Skip Grammar** o $scoreVec = v_{wAverage}^T \cdot W_2$ per **CBOW**.
2. **Multi-class classification** attraverso **softmax**, $softmax(scoreVec) = \frac{e^{scoreVec}}{\sum_{i=1}^{V} e^{scoreVec_i}}$

Dove $V = sizevocabulary$, $scoreVec_i$ è l'i-esimo componente di $scoreVec$ e e^{vector} rappresenta l'esponenziale element by element.



Word2Vec - Parameters Learning - Loss

Il **loss** viene definito come **una media** dei **loss relativi** ad ogni **training sample**, nel nostro caso un training sample è una **parola nel corpus**.

$$L = \frac{1}{M} \cdot \sum_{m=1}^{M} L_m$$

L'espressione del loss in questi termini serve solo a **visualizzare** il suo valore durante le **iterazioni dell'algoritmo**.
Un **epoch** (iterazione) della procedura di training consiste nell'effettuare per ogni **training sample** la procedura di **forward e back propagation**.
Per ogni sample in un **epoch** vengono quindi aggiornate tutte le **weights** del modello.
Ad ogni **epoch** la massimizzazione del **loss** sarà effettuata considerando il loss L_m relativo a sample che sta attraversando il network.

Word2Vec - Parameters Learning

Cominciamo considerando il caso di **CBOW** in cui abbiamo una sola parola nel contesto, vale a dire il modello dovrà predire la **focus word** data **una** parola nel **contesto**.

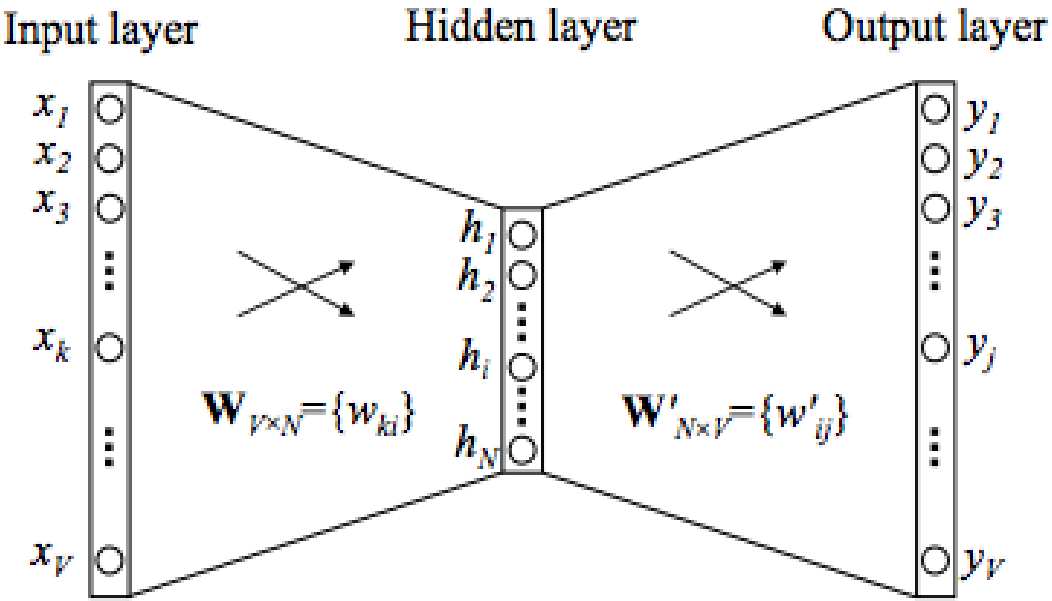


Figure 1: A simple CBOW model with only one word in the context

In questa figura $W = W_1$ e $W' = W_2$.

Word2Vec - Parameters Learning - CBOW - L_m

Considerando il **training sample m** che attraversa il network e ipotizzando w_{m-1} l'unica parola nel contesto di w_m;

$$L_m = \log p(w_m|w_{context}) = \log \left(\frac{exp(v'^T_{w_m} \cdot v^T_{w_{context}})}{\sum_{i=1}^{V} exp(v'^T_{w_i} \cdot v^T_{w_{context}})} \right) =$$

$$= \log \left(\frac{exp(v'^T_{w_m} \cdot h)}{\sum_{i=1}^{V} exp(v'^T_{w_i} \cdot h)} \right) = \log \left(\frac{exp(u_m)}{\sum_{i=1}^{V} exp(u_i)} \right) =$$

$$= \log \left(exp(u_m) \right) - \log \left(\sum_{i=1}^{V} exp(u_i) \right) =$$

$$= u_m - \log \left(\sum_{i=1}^{V} exp(u_i) \right) = -E$$

Quindi l'errore commesso per **UN SOLO TRAINING SAMPLE** è:

$$E = -\log p(w_m|w_{m-1})$$

Word2Vec - Parameters Learning - Forward Propagation

Chiamiamo:

- u_i con $i \in 1, 2, .., V$ lo score calcolato per la parola i.
- h_i con $i \in 1, 2, .., N$ l'output del i-th neurone dell'hidden layer.
- y_i con $i \in 1, 2, .., V$ l'output del i-th neurone dell'output layer.

Considerando un **OHE** vector in input $x_{context}$ di dimensioni $[V, 1]$ con il k-th elemento $x_k \neq 0$, definiamo:

- $h = \sigma(W^T \cdot x_{context}) = W^T \cdot x_{context} = v^T_{w_{context}}$, vale a dire che la funzione di attivazione per il primo layer è la funzione identità (**lookup table**).
- $h_i = \sum_{k=1}^{V} x_k \cdot w_{ki}$
- $u_j = v'^T_{w_j} \cdot h$
- $y_j = p(w_j | w_{context}) = \frac{exp(u_j)}{\sum_{i=1}^{V} exp(u_i)}$, la probabilità a posteriori che $w_j \in V$ sia la parola campionata nel contesto di $w_{context}$.

Word2Vec - Parameters Learning - W' (W_2)

$$\frac{\partial E}{\partial u_j} = \frac{\partial \left(\log \left(\sum_{i=1}^{V} exp(u_i) \right) - u_m \right)}{\partial u_j} =$$

$$= \frac{\partial \left(\log \left(\sum_{i=1}^{V} exp(u_i) \right) \right)}{\partial u_j} - \frac{\partial (u_m)}{\partial u_j} =$$

$$= \frac{exp(u_j)}{\sum_{i=1}^{V} exp(u_i)} - t_j =$$

$$= y_j - t_j = e_j$$

$$\text{dove } t_j = \begin{cases} 1 & \text{if } j = m \\ 0 & \text{if } j \neq m \end{cases}$$

$$\frac{\partial u_j}{\partial w'_{ij}} = \frac{\partial \left(v'^{T}_{w_j} \cdot h \right)}{\partial w'_{ij}} = h_i$$

$$\frac{\partial E}{\partial w'_{ij}} = \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial w'_{ij}} = e_j \cdot h_i$$

Word2Vec - Parameters Learning - W' (W_2)

L'aggiornamento dei parametri quindi risulta:

$$w_{ij}'^{(t+1)} = w_{ij}'^{(t)} - \eta \cdot h_i \cdot e_j \qquad i \in 1, 2, .., N \wedge j \in 1, 2, .., V$$

$$v_{w_j}'^{(t+1)} = v_{w_j}'^{(t)} - \eta \cdot h \cdot e_j \qquad j \in 1, 2, .., V$$

Di conseguenza ad ogni ciclo vengono aggiornati tutti i pesi della matrice W_1.

Word2Vec - Parameters Learning - W (W_1)

$$\frac{\partial E}{\partial h_i} = \sum_{j=1}^{V} \left(\frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial h_i} \right) = \sum_{j=1}^{V} \left(e_j \cdot \frac{\partial u_j}{\partial h_i} \right) =$$

$$= \sum_{j=1}^{V} \left(e_j \cdot w'_{ij} \right) = e \cdot (W_{2_i})^T$$

Dove W_{2_i} è la i-esima riga di W_2 ($[1, V]$), che trasposta da un vettore $[V, 1]$, il quale, moltiplicato per un vettore $[1, V]$ da uno scalare.

$$\frac{\partial h_i}{\partial w_{ki}} = x_k$$

$$\frac{\partial L_m}{\partial w_{ki}} = \frac{\partial L_m}{\partial h_i} \cdot \frac{\partial h_i}{\partial w_{ki}} = x_k \cdot e \cdot (W_{2_i})^T$$

Word2Vec - Parameters Learning - W (W_1)

L'aggiornamento dei parametri quindi risulta:

- $w_{ki}^{(t+1)} = w_{ki}^{(t)} - \eta \cdot x_k \cdot e \cdot (W_{2_i})^T$, con $k \in 1, 2, .., V$ e $i \in 1, 2, .., N$
- $W_1^{(t+1)} = W_1^{(t)} - \eta \cdot x \cdot \left(e \cdot W_2^T\right)$

Dove $x \cdot \left(e \cdot W_2^T\right)$ è di dimensionalità $[V, N]$.
Considerando poi che x è un **OHE vector**, solo un coefficiente di x sarà $\neq 0$. Di conseguenza questa operazione di aggiornamento lascia inalterate tutte le **input representation** della matrice W_1 tranne quella corrispondente all'indice $x_k \neq 0$ di x.
Quindi l'unico aggiornamento che avviene è:

- $v_{w_{context}}^{(t+1)} = v_{w_{context}}^{(t)} + \eta \cdot \left(e \cdot W_2^T\right)$

Di conseguenza vediamo una delle particolarità di **word2vec**, vale a dire che **per ogni parola in input l'aggiornamento di W_1 si limita solo al word vector relativo alla input word, in questo caso di** $v_{w_{context}}$.

Word2Vec - Parameters Learning - Batch GD

Invece di considerare un solo sample per volta possiamo anche considerare un **batch** X di M samples:

- $X =$ una **OHE** riga per ogni **input sample** in M, $[M, V]$
- $T =$ una **OHE** riga per ogni **target sample** in M, $[M, V]$

La **forward pass** sarà:

- $H = X \cdot W_1$ $[M, N]$
- $U = H \cdot W_2$, $[M, V]$
- $Y = softmax(U)$, $[M, V]$, element-wise

La **back pass** sarà:

- $\frac{\partial E}{\partial U} = \frac{1}{M} \cdot (Y - T)$, $[M, V]$
- $\frac{\partial E}{\partial W_2} = H^T \cdot \frac{\partial E}{\partial U}$, $[N, V]$
- $\frac{\partial E}{\partial H} = \frac{\partial E}{\partial U} \cdot W_2^T$, $[M, N]$
- $\frac{\partial E}{\partial W_1} = X^T \cdot \frac{\partial E}{\partial H}$, $[V, N]$

Word2Vec - Parameters Learning - Complete CBOW

Considerando il modello **CBOW** con multiple parole nel contesto della parola w_m avremo pochi cambiamenti rispetto al modello appena spiegato.

- $h = \frac{1}{C} \cdot W_1^T \cdot (x_{m-C} + ... + x_{m-2} + x_{m-1} + x_{m+1} + x_{m+2} + ... + x_{m+C})$

Di conseguenza il vettore h diventa uno **pseudo-OHE**, cioè un vettore che ha dimensione del vocabolario e un numero di 1 pari al numero di parole nel contesto.
La procedura si svolge nello stesso modo, tranne per quanto riguarda l'aggiornamento dei pesi della matrice W_1, infatti ora aggiorno un numero di **input representation** pari al numero della parole nel contesto:

- $v_{w_{m-c}}^{(t+1)} = v_{w_{m-c}}^{(t)} + \eta \cdot \frac{1}{C} \cdot \left(e \cdot W_2^T\right)$, con $c \in C, ..., -C$ e $c \neq 0$

Per quanto riguarda W_2 invece non cambia niente:

- $v_{w_j}^{\prime(t+1)} = v_{w_j}^{\prime(t)} + \eta \cdot h \cdot e_j$, con $j \in 1, 2, .., V$

Word2Vec - Parameters Learning - Skip Gram - L_m

Considerando il **training sample m** che attraversa il network e ipotizzando w_{m-1} l'unica parola nel contesto di w_m

$$L_m = \sum_{-c<j<c, j\neq 0} \log p(w_{m+j}|w_m) = \sum_{-c<j<c, j\neq 0} \frac{exp(v'^T_{w_{m+j}} \cdot v^T_{wm})}{\sum_{i=1}^{V} exp(v'^T_{w_i} \cdot v^T_{w_m})}$$

$$= \sum_{-c<j<c, j\neq 0} \frac{exp(u_{m+j})}{\sum_{i=1}^{V} exp(u_i)} =$$

$$= \sum_{-c<j<c, j\neq 0} \log exp(u_{m+j}) - \sum_{-c<j<c, j\neq 0} \log \sum_{i=1}^{V} exp(u_i) =$$

$$= \sum_{-c<j<c, j\neq 0} u_{m+j} - C \cdot \log \sum_{i=1}^{V} exp(u_i) = -E$$

Quindi l'errore commesso per **UN SOLO TRAINING SAMPLE** è:

$$E = -\sum_{-c<j<c, j\neq 0} \log p(w_{m+j}|w_m)$$

Word2Vec - Parameters Learning - Skip Gram - Forward Propagation

Chiamiamo:

- $u_{c,j'} = u_{j'} = v'^{T}_{w_{j'}} \cdot h$, per ogni parola nel contesto di v_{w_m}, infatti gli output panels condividono la stess matrice di weights W_2.
- $h = v^{T}_{w_m}$

Word2Vec - Parameters Learning -Skip Gram - W' (W_2)

$$\frac{\partial E}{\partial u_{m+c,j'}} = \frac{\partial \left(C \cdot \log \sum_{i=1}^{V} exp(u_i) - \sum_{-c<j<c, j \neq 0} u_{m+j} \right)}{\partial u_{m+c,j'}} =$$

$$= \frac{\partial \left(C \cdot \log \left(\sum_{i=1}^{V} exp(u_i) \right) \right)}{\partial u_{m+c,j'}} - \frac{\partial \left(\sum_{-c<j<c, j \neq 0} u_{m+j} \right)}{\partial u_{m+c,j'}} =$$

$$= \frac{\partial \left(C \cdot \log \left(\sum_{i=1}^{V} exp(u_i) \right) \right)}{\partial u_{m+c,j'}} - t_{m+c,j'} =$$

Word2Vec - Fake task

L'aspetto interessante di questa tecnica è che, dopo il training, **la rete non verrà utilizzata per la task su cui è stata addestrata**. Il training serve solo per aggiornare e migliorare i pesi delle matrici W_1 e W_2.

- Quello che ci interessa ottenere dopo la fase di training sono le weights della matrice W_1 che rappresenteranno gli **word vectors** per le parole del vocabolario (row-wise).

Word2Vec - Improvements

Il modello presentato ha dei problemi nel momento in cui si vuole applicare la backpropagation per l'aggiornamento delle weights. Infatti abbiamo due matrici da aggiornare che hanno in media 1M di weights ciascuna. Questo risulta in alcune problematicità, come ad esempio:

- time complexity.
- abbiamo bisogno di molti training data per evitare over-fitting.

Word2Vec - Improvements

Nel secondo paper gli autori di questa tecnica hanno proposto 3 miglioramenti:

- Phrases identification.
- Subsampling common words.
- Modificare il training objective usando la tecnica del negative sampling che porta ogni training example ad aggiornare solo una piccola percentuale di weights.

Questo procedimento ha anche significativamente migliorato le performance del modello.

Word2Vec - Improvements - Negative Sampling

Ci soffermiamo sulla terza tecnica perchè è la più interessante.

- W_1**:** Aggiorno solo il **weight vector** relativo alla input word, ma questo avviene a prescindere dal **"negative subsampling"**.
- W_2**:** Invece di aggiornare tutti gli **weight vectors** della matrice W_2 ad ogni training sample processato, prevede di selezionare un sottoinsieme di "negative words" da aggiornare.
  In questo contesto "negative word" corrisponde ad una parola per cui il target vector presenta uno 0.
  Solitamente il numero di negative selections che vengono fatte sono da 5 a 20.

Word2Vec - Improvements - Negative Sampling

Per selezionare gli **word vectors della matrice** W_2 da aggiornare consideriamo solo quelli riferiti alle parole più frequenti nel testo. La probabilità per una parola di essere campionata è:

- $p(w_i) = \frac{f(w_i)^{\frac{3}{4}}}{\sum_{j=0}^{V} f(w_j)^{\frac{3}{4}}}$

In questo modo possiamo **ridurre drasticamente il numero di weights che vengono aggiornate ad ogni training sample**, indicativamente con 3M di weights e 1800 aggiornamenti con "negative sampling" aggiorniamo lo 0.06% delle weights.

Word2Vec - Giustification

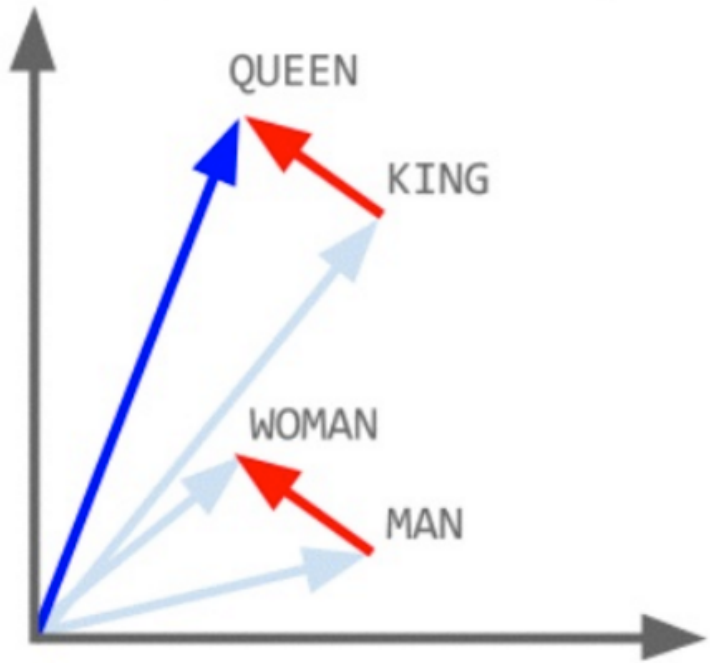
Se due parole vengono usate in contesti simili, allora il modello deve generare risultati molto simili per queste due parole.
Un modo che la NN ha per fare predizioni simili per queste due parole è utilizzare due **word vectors** simili.
La rete così risulta motivata a imparare simili **word vectors** per queste due parole.

Word2Vec - Example

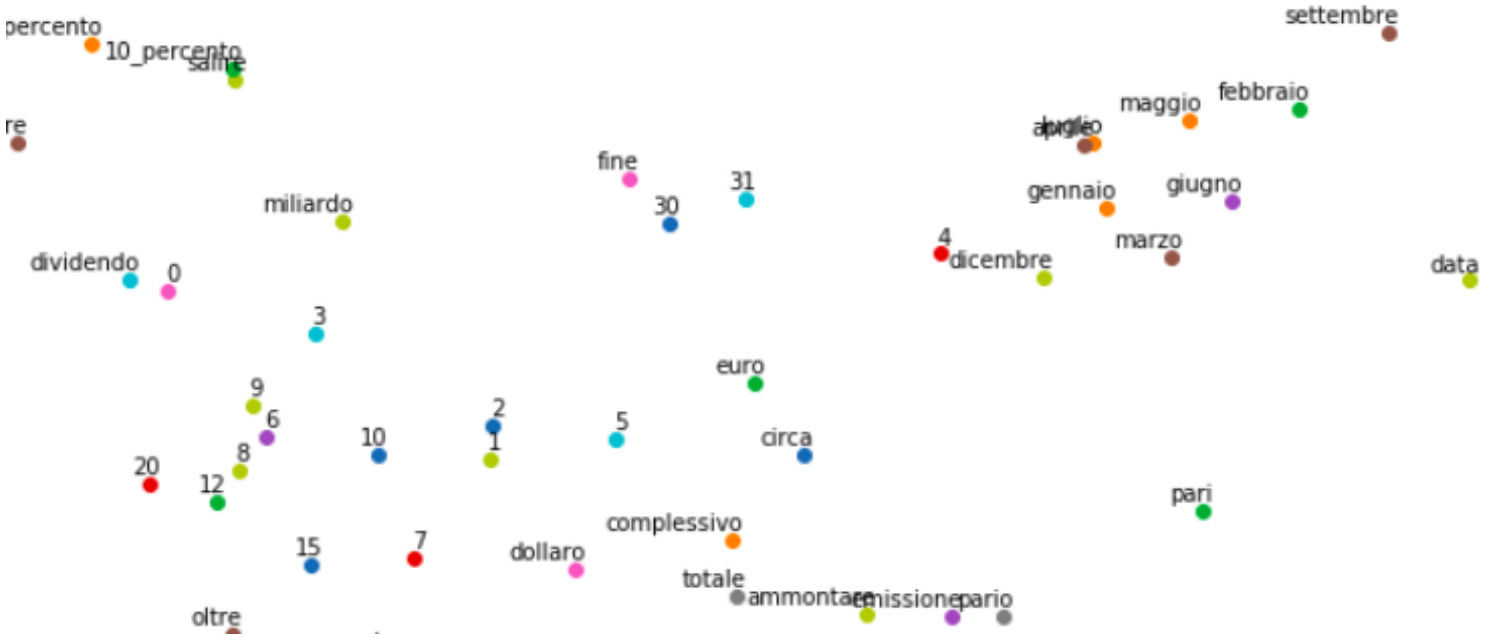
Gli **word embeddings** ottenuti tramite **word2vec** hanno la particolare qualità di catturare le relazioni tra i termini.
Per esempio il risultato della espressione:

$$vector(king) - vector(man) + vector(woman)$$

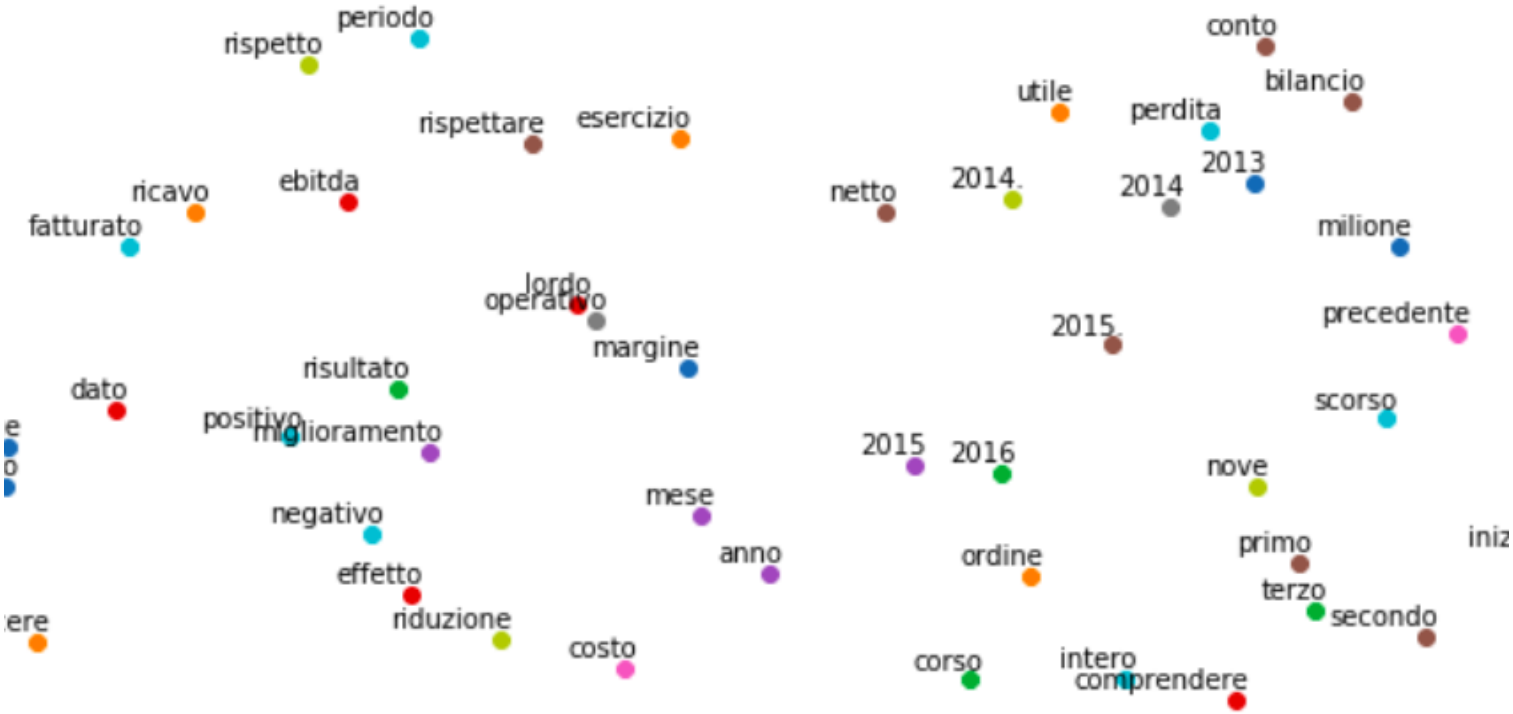
è un vettore **vicino** a $vector(queen)$.



RadioCor e Sole 24 Ore - Dettaglio 1



RadioCor e Sole 24 Ore - Dettaglio 2



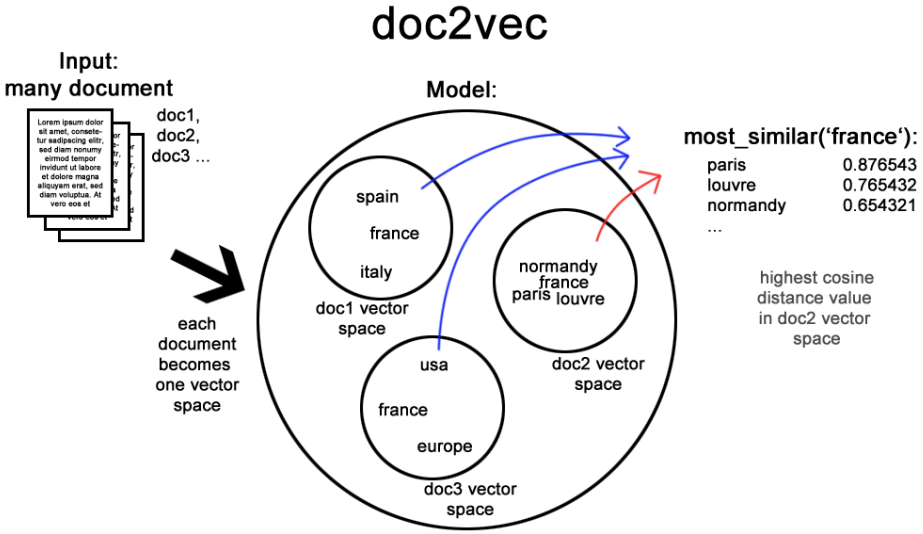
Outline

Introduzione

Doc2Vec è l'estensione di **Word2Vec**, che permette di calcolare similarità tra **documenti**, dato un corpus di documenti.
Per questo task è basato anch'esso su una **two-layers neural network** che viene trainata sul corpus di documenti.

- **Input: corpus** di testi di lunghezza arbitraria
- **Output: vector-space**

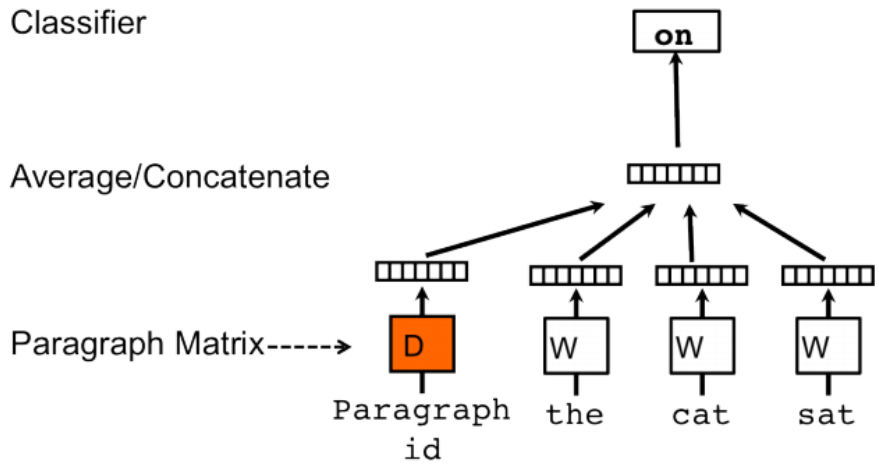
La creazione di tale famiglia di modelli data dalla rappresentazione con **Paragraph Vector** che permette di lavorare con documenti, paragrafi o frasi di **lunghezza variabile**.



Doc2Vec - CBOW Model

Consideriamo l'architettura del **CBOW**, presente già in **Word2Vec**.
La struttura è la seguente:

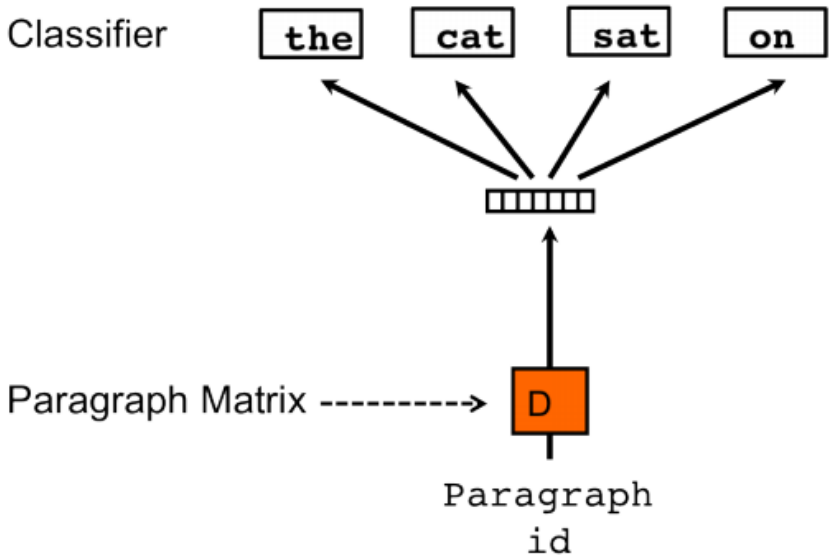
- L'**input** prevede $w_1, w_2, ...w_m$ parole del paragrafo, dove $\mathrm{M} = \mathit{windowsize}$ e il **Paragraph id** che rappresenta lo specifico paragrafo.
- L'**output** è la parola del contesto w_{m+j}



Doc2Vec - Skip Grammar

Per quanto riguarda l'architettura del **Skip Gram**, presente già in **Word2Vec**.
La struttura è la seguente:

- L'**input** prevede il **Paragraph id** che rappresenta lo specifico paragrafo.
- L'**output** sono $w_1, w_2, ...w_m$, parole del paragrafo, dove $\mathrm{M} = \textit{windowsize}$



Paragraph Vector

Quindi il vettore di ogni parola **è condiviso** tra ogni paragrafo.
Mentre il **Paragraph id**, cioé la colonna corrispondente al Paragraph nella **Paragraph Matrix**, **identifica univocamente** ognuno dei paragraph.
Il **Paragraph id** costituisce una sorta di **memoria** del topic del paragraph.
Come accade in Word2Vec, tutti i vettori di input sono **averaged** o **concatenated**.

Doc2Vec - Risultati

Confronto delle performances del Paragraph Vector sull'IMDB dataset.
Preso da Distributed Representations of Sentences and Documents

Model	Error rate
BoW (bnc) (Maas et al., 2011)	12.20 %
BoW (bΔt'c) (Maas et al., 2011)	11.77%
LDA (Maas et al., 2011)	32.58%
Full+BoW (Maas et al., 2011)	11.67%
Full+Unlabeled+BoW (Maas et al., 2011)	11.11%
WRRBM (Dahl et al., 2012)	12.58%
WRRBM + BoW (bnc) (Dahl et al., 2012)	10.77%
MNB-uni (Wang & Manning, 2012)	16.45%
MNB-bi (Wang & Manning, 2012)	13.41%
SVM-uni (Wang & Manning, 2012)	13.05%
SVM-bi (Wang & Manning, 2012)	10.84%
NBSVM-uni (Wang & Manning, 2012)	11.71%
NBSVM-bi (Wang & Manning, 2012)	8.78%
Paragraph Vector	**7.42%**

References

1. **Efficient Estimation of Word Representations in Vector Space:** presentazione Skip Gram e CBOW models
2. **Distributed Representations of Words and Phrases and their Compositionality:** Word2Vec
3. **Paragraph Vectors**
4. **Document Embedding with Paragraph Vectors:** Doc2Vec