

UNIVERSITÀ DEGLI STUDI DI  
MILANO-BICOCCA

BIG DATA IN PUBLIC AND SOCIAL SERVICES  
FINAL PROJECT

---

# Un sistema di auto-completamento semantico di skills

---

*Authors:*

Simone D'Amico - 850369  
s.damico4@campus.unimib.it

*Tutor:*

Andrea Seveso  
a.seveso6@campus.unimib.it





## Abstract

Negli questi anni si è assistito ad un forte incremento della nascita di nuove professioni che coinvolgono e richiedono un numero sempre maggiore di conoscenze e competenze. La necessità di mettere in comunicazione domanda e offerta passa anche dalla corretta individuazione delle competenze encessarie per svolgere un determinato lavoro. Assumono sempre maggiore inportanza aree di studio come la Labor Market Intelligence (LMI). L'obiettivo di LMI è supportare le attività e le politiche decisionali (ad esempio, monitoraggio in tempo reale delle offerte di lavoro online (OJV) in tutti i paesi, previsione delle competenze richieste nei posti vacanti, confronto di mercati del lavoro simili a livello transfrontaliero, ecc.)[1]. In questo lavoro si descrive lo sviluppo di un sistema di suggeriemto di skills dalla tassonomia standard europea delle occupazioni e delle competenze (E.S.C.O.)

## Introduzione

Lo scopo del progetto è cercare di trovare risposta ad alcune domande che catturano sempre maggior interesse in settori come la data science e lo studio di NLP: *(Q1) Quali strumenti possono essere utilizzati per produrre un sistema che possa suggerire parole che siano, non solo sinatticamente ma anche semanticamente simili?* e ancora *(Q2) In che modo tale sistema possa calcolare la similarità tra le parole e suggerire quelle più adatte ad un particolare contesi?*



In questo report si descrivono le fasi di sviluppo di un sistema di auto-completamento di parole che sia in grado di suggerire quelle skills maggiormente simili, dal punto di vista semantico, a quelle inserite dall'utente.


Si cercherà di rispondere a tali domande di ricerca partenda da studi precedenti per realizzare, in prima istanza un modello e una metrica per il calcolo delle similarità e, successivamente, un sistema che implementi tale modello.

Nella sezione 1 si descrive il lavoro intorno allo sviluppo del sistema NEO da cui, in parte, nasce e si sviluppa l'idea di questo sistema di suggerimenti, sebbene il lavoro intorno a NEO sia molto ampio, in questa sede ci si concentra sulle modalità che hanno portato all'individuazione delle skills e dei relativi vettori di embeddings. Nella sezione 2 si definisce il modello di suggerimento in modo formale descrivendo l'analisi che ha portato poi alla definizione della metrica utilizzata per fornire i suggerimenti. Nella sezione 3 si descrive l'architettura del sistema e dei due moduli che la compongono. Nella sezione 4 NON LO SO e infine nella sezione 6 si presentano le conclusioni e i futuri sviluppi legati al sistema.

# 1 Related work

Il progetto si basa su sistema NEO e il relativo paper *NEO: A Tool for Taxonomy Enrichment with New Emerging Occupations*[2]; il cui scopo è quello di realizzare un sistema europeo per raccogliere e classificare annunci di lavoro online (OJVs) per l'intera UE e le sue 32 lingue dell'Unione[3] attraverso l'apprendimento automatico basandosi sulla tassonomia E.S.C.O. (European Skills, Competences, Qualifications and Occupations). La classificazione E.S.C.O. identifica e classifica skills, competenze, qualifiche e occupazioni rilevanti per il mercato del lavoro europeo e l'istruzione e la formazione[4]. E.S.C.O. è stato sviluppato dalla Commissione Europea dal 2010 e fornisce descrizioni di 2942 professioni e 13.485 competenze legate a queste occupazioni, tradotte in 27 lingue (tutte le lingue ufficiali dell'UE più islandese, norvegese e arabo)[4].

 lavora Il lavoro descritto nel paper vuole realizzare un sistema che (1) impari i word embeddings di concetti e entità delle tassonomie preservando le relazioni tassonomiche, (2) suggerisca nuove entità per E.S.C.O. estratte da un corpus testuale e (3) valuti, tramite misure G.A.S.C., la loro idoneità come entità di differenti concetti tassonomici[2]. 

Per la realizzazione del sistema di auto-completamento ci si concentra  unicamente sul primo di questi obiettivi: la definizione del modello con cui sono stati imparati i vettori di embeddings associati alle skills della classificazione E.S.C.O. da usare come suggerimenti.

Il corpus utilizzato contiene 2,119,025 annunci di lavoro pubblicati nel Regno Unito durante il 2018, con riferimento a concetti E.S.C.O. nel settore ICT. Ogni annuncio è stato preprocessato eseguendo: (1) tokenization, (2) riduzione lower case, (3) rimozione punteggiatura e stopwords, (4) individuazione degli n-grams ottenendo l'insieme delle skills utilizzate nel sistema di auto completamento.

Nella fase sperimentale sono stati impiegati e valutati tre delle architetture più importanti per imparare word embedding da grandi corpora di testi: GloVe, Word2Vec, and FastText. Ognuno dei tre algoritmi è stato utilizzato con diversi parametri per generare un totale di 260 modelli. In conclusione, il modello con le migliori performance è risultato avere l'architettura FastText con algoritmo CBOW e 300 componenti per vettore. L'utilizzo di FastText come generatore di embeddings permette di avere a disposizione, non solo i vettori associati alle skills ma anche gli embeddings delle varie loro sottoparole. Questo permette di gestire errori di typo nell'inserimento di parole e poter calcolare la similarità in modo incrementale ad ogni nuovo input dell'utente.

## 2 Definizione del problema

In questa sezione si descrive il modello su cui è stato realizzato il sistema di auto-completamento da un punto di vista formale, si introducono i dati a disposizione, la modellazione del problema proposto e la metrica di valutazione utilizzata per individuare i suggerimenti più adatti.

Dal lavoro proposto da Giabelli et al[2] si sono ottenute un totale di 2319 skills, che possono essere singole parole o espressioni, inoltre si hanno a disposizione vettori composti da 50 componenti che rappresentano non solo le skills ma anche loro sotto-stringhe, per un totale di 113656 vettori.

### 2.1 Formalizzazione del problema

Nella formalizzazione del problema si identifica:

- L'insieme delle skills  $S$  definite nella classificazione E.S.C.O.
- L'input dell'utente  $i$  che può essere l'esatta skill o una sua sotto-stringa.
- Il contesto  $C = \{c_1, \dots, c_n\}$  delle skill già inserite dall'utente.

Per ogni input dell'utente si considera la stringa inserita fino a quel momento, si calcola poi la similarità del coseno tra  $i$  e ogni skills ottenendo delle coppie composte dalla skill e dalla sua similarità con l'input, si escludono le skills già inserite, cioè quelle contenute nel contesto:

$$A = \{(s, \text{sim}(i, s)) \mid s \in S \ \& \ s \notin C\} \quad (1)$$

Successivamente si considera il contesto, e quindi, per ogni skill già inserita, si calcola la sua similarità con le skill candidate come suggerimento:

$$B = \{(s, \text{sim}_{avg}(s, C)) \mid s \in S \ \& \ s \notin C\} \quad (2)$$

Dove  $\text{sim}_{avg}(s, C)$  indica la media delle similarità del coseno della skill  $s$  con le skills nel contesto  $C$  ed è calcolata per ogni skill candidata:

$$\text{sim}_{avg}(s, C) = \frac{\sum_{c \in C} \text{sim}(s, c)}{|C|} \quad (3)$$

Nell'insieme  $A$  viene calcolata quindi la similarità tra l'input dell'utente e le skills a disposizione mentre nell'insieme  $B$  si calcola la similarità tra le skill da poter suggerire e quelle già inserite.

## 2.2 Metrica di valutazione delle similarità

Calcolate le similarità sia considerando l'input che le skills già inserite, si passa poi ad identificare la skill maggiormente correlata da poter suggerire all'utente. Si utilizza una combinazione lineare pesata con un parametro  $\alpha \in [0, 1]$ , per ogni skill si calcola:

$$\begin{aligned} \forall s \in S \ \& \ s \notin C \\ \text{sim}_s &= \alpha \cdot \text{sim}_A + (1 - \alpha) \cdot \text{sim}_B & \begin{aligned} (s, \text{sim}_A) &\in A \\ (s, \text{sim}_B) &\in B \end{aligned} \end{aligned} \quad (4)$$

Utilizzando l'equazione (4) si calcolano le similarità finali di tutte le skills e si suggerisce quella con valore di similarità massimo.

L'utilizzo del parametro  $\alpha$  permette al modello di essere flessibile e, a seconda dello scenario di utilizzo, dare maggiore importanza all'input corrente dell'utente o alle skills già inserite: per valori di  $\alpha$  vicini a 1 aumenta l'importanza della skill che l'utente sta inserendo mentre per valori di  $\alpha$  vicini a 0 il contesto delle skills già inserite assume sempre maggiore importanza nella scelta del suggerimento.

## 3 Implementazione del sistema

Il sistema è quindi in grado di fornire due tipi di suggerimenti: consiglia le skills più correlate sia con il contesto che con l'input dell'utente come descritto nella sezione precedente, inoltre, fornisce suggerimenti ignorando l'input dell'utente e considerando solo le skills complete del contesto.

Il sistema è stato realizzato utilizzando il linguaggio di programmazione *Python* e l'applicazione open-source *Jupyter Notebook*.

Il sistema si compone di due componenti principali realizzate come classi *Python*: il modulo *AutoCompleteManager* implementa il modello di auto-completamento descritto nella sezione 3, mentre il modulo *FormUI* gestisce una piccola interfaccia utente con la quale interagire con il sistema. L'interazione avviene tramite un'area di testo nella quale inserire le skills di interesse, i suggerimenti vengono mostrati come bottoni, nel momento della pressione dello specifico bottone, la relativa skill viene aggiunta all'area di testo.

Per migliorare l'esperienza utente, i suggerimenti sono mostrati a partire dal quarto carattere inserito.

Si è cercato di implementare il sistema in modo che sia più flessibile possibile e facilmente riutilizzato in contesti diversi, questo si è tradotto nella scelta dei parametri di inizializzazione del sistema. Quando si effettua il deploy del sistema possono essere scelti alcuni parametri, come il numero di suggerimenti

menti da mostrare (di default 4) e il valore di  $\alpha$  usato per la pesatura delle similarità. Il sistema può essere usato anche in diversi contesti fornendo una lista di parole e relativi embeddings in base al settore del suo utilizzo.

### 3.1 Modulo AutoCompleteManager

Il modulo è stato realizzato come classe *Python* con diverse funzioni, ognuna dedicata ad uno specifico compito:

- Il costruttore della classe si occupa di inizializzare una serie di parametri come la lista delle skills, i vettori di FastText, il parametro  $\alpha$  per il calcolo della similarità e crea un'istanza della classe *FormUI* da usare come interfaccia.
- La funzione `get_skills_input_similarity` inizializza l'insieme  $A$  definito nell'equazione (1). Utilizza un dizionario le cui chiavi sono le skills e i rispettivi valori sono le similarità delle skills con l'input dell'utente inserito fino a quel momento.
- La funzione `get_skills_context_similarity` inizializza l'insieme  $B$  definito nell'equazione (2). Utilizza un dizionario le cui chiavi sono le skills e i rispettivi valori sono le similarità delle skills con il contesto.
- La funzione `get_best_similarity_skill` calcola la similarità totale come specificato in (4) richiamando le due funzioni precedenti.




---

**Algorithm 1:** *get\_best\_similarity\_skill*: calcola le similarità secondo l'eq. (4)

---

**Input:**  $\alpha$ , *num\_suggests*: numero di suggerimenti da mostrare, *skills\_list*: le skills E.S.C.O.



**Output:** Le skill più simili sia al contest che all'input con i relativi valori di similarità

```

1  $sim_A \leftarrow get\_skills\_input\_similarity(new\_input, context)$ 
2  $sim_B \leftarrow get\_skills\_context\_similarity(new\_input, context)$ 
3  $res \leftarrow dict()$ 
4 for  $s$  in  $sim_A.keys()$  do
5    $res[s] = \alpha * sim_A[s] + (1 - \alpha) * sim_B[s]$ 
6  $most\_similar \leftarrow sorted(res)[0 : num\_suggests]$ 
7 return  $most\_similar$ 

```

---


- `suggest_interest_skill` consiglia le skills maggiormente correlate al contesto, ignorando l'input corrente dell'utente.

- La funzione `add_skill` è eseguita ad ogni pressione del bottone e inserisce la skill associata al bottone cliccato nell'area di testo, inoltre resetta i bottoni permettendo di mostrare successivamente nuovi suggerimenti.
- La funzione `suggests_manager` è la funzione di listener associata all'area di testo nel quale l'utente inserisce le skills, viene invocata ogni volta che si inserisce un nuovo carattere nell'area di testo. Individua i migliori suggerimenti richiamando le varie funzioni.

---



**Algorithm 2:** *suggests\_manager*: calcola le similarità ad ogni nuovo input e mostra i suggerimenti più adatti

---

**Input:** *num\_suggests*, *lista\_unique*, *primary\_key* 

**Output:** Le proprietà definite con le API generiche

```

1 new_input ← parola inserita fino a quel momento
2 context ← lista delle skills già inserite
3 if context == [] then
    // si sta inserendo la prima skill 
4     simA ← get_skills_input_similarity(new_input, context)
5     suggests ← sorted(simA)[0 : num_suggests]
6     for s in suggests do
7         Crea e visualizza il bottone con descrizione s e la sua imilarità
            con new_input
8 else
    // si sono già inserite altre skills
9     bestsimilarity ← get_best_similarity_skill(new_input, context) 
10    suggests ← sorted(simA)[0 : num_suggests]
11    for s in suggests do
12        Crea e visualizza il bottone con descrizione s e la sua imilarità
            con new_input
13 return dddd

```

---

- `show_form` richiama l'omonima funzione nel modulo *FormUI* che permette di visualizzare l'interfaccia.

### 3.2 Modulo FormUI

Il modulo *FormUI* si occupa di inizializzare il form che compone di fatto l'interfaccia del sistema. Come mostrato in figura 3.2, il form è composto da tre sezioni distinte: la text area, con cui l'utente inserisce le skills di interesse e in cui si auto-completa l'input secondo il suggerimento scelto, la

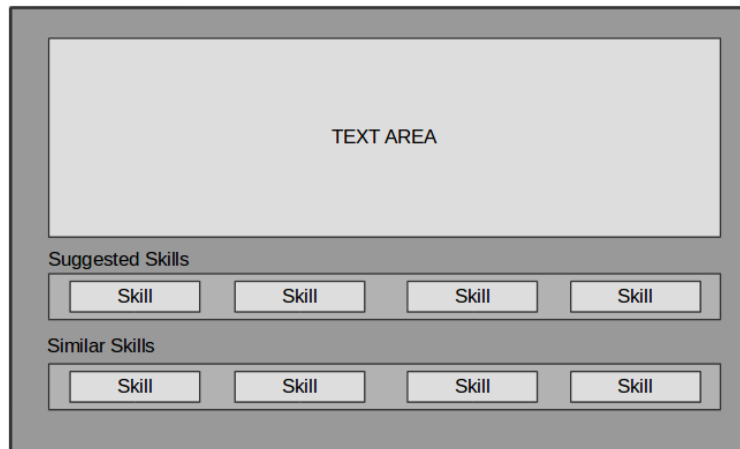


Figure 1: Struttura dell'interfaccia utente.

sezione *Suggested Skills* in cui sono presenti i bottoni etichettati con il nome della skill e la sua similarità; in questa sezione i suggerimenti sono calcolati considerando sia l'input che il contesto tramite la loro combinazione lineare e, infine, la sezione *Similar Skills* in, analogamente alla precedente, sono mostrati i bottoni associati alle skills che sono più simili solo al contesto. La classe Python implementa una serie di funzioni:

- Il costruttore inizializza le tre aree sopra descritte associandovi anche un layout per la loro posizione.
- La funzione `show_form` mostra l'intero form.
- La funzione `init_button` con cui creare i bottoni dei suggerimenti, alla funzione vengono passati il testo da mostrare come descrizione del bottone e come tooltip al passaggio del mouse su di esso.
- Le funzioni `close_suggest_buttons` e `close_suggest_buttons_context` rimuovono i bottoni, rispettivamente della sezione *Suggested Skills* e *Similar Skills*, successivamente alla scelte dell'utente permettendo la visualizzazione di bottoni con nuovi suggerimenti.

Nella figura 3.2 si può vedere l'interfaccia utente in attività durante l'interazione, sono presenti le tre sezioni descritte con i relativi suggerimenti.



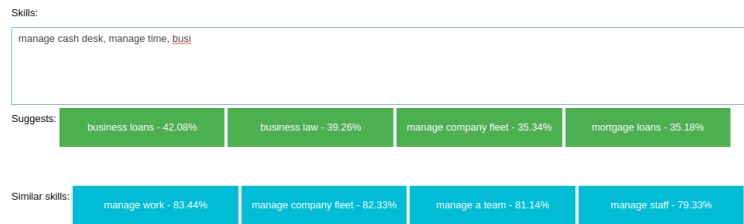


Figure 2: Interfaccia utente durante l'interazione.

## 4 Analisi delle prestazioni

## 5 Futuri sviluppi

Le informazioni che il sistema utilizza per fornire suggerimenti sono solo quelle relative ai vettori di embedding delle skills identificate dalla classificazione E.S.C.O., non si prende in considerazione quindi la gerarchia a più livelli della classificazione. Per rendere i suggerimenti più efficaci si potrebbe inglobare nella misura della similarità mostrata in (4) anche una misura di distanza tra le skills nella gerarchia come descritto in Giabelli et al[1].

Un'ulteriore ampliamento di questo lavoro potrebbe considerare le informazioni legate direttamente agli annunci di lavoro come l'anno o l'area geografica di pubblicazione, quindi ricalcolare gli embeddings considerando solo gli annunci filtrati per queste caratteristiche.

Ulteriore oggetto di studio può essere il comportamento del sistema se si utilizzano differenti metriche di similarità, ad ora quella che viene calcolata è la similarità del coseno, ma possono essere impiegate anche altre misure come la distanza Euclidea o il dot product.

## 6 Conclusioni

Il sistema presentato è in grado di fornire suggerimenti in modo dinamico secondo gli interessi dell'utente aiutandolo sia nel completare una skill che si sta inserendo e sia consigliando le skills più simili in assoluto rispetto a quelle già inserite.

Per rispondere alla prima domanda di ricerca, *(Q1) Quali strumenti possono essere utilizzati per produrre un sistema che possa suggerire parole che siano, non solo sinatticamente ma anche semanticamente simili*, si è scelto di utilizzare tecniche di word embedding per ottenere una rappresentazione

vettoriale delle parole attraverso l'algoritmo di FastText. Grazie a questo si è realizzato un modello con cui calcolare le similarità delle varie skills utilizzando la similarità del coseno e la metrica generale descritta in (4) per poter identificare i suggerimenti più adatti da proporre all'utente. In questo modo si è stati in grado di rispondere anche alla seconda domanda di ricerca: *(Q2) In che modo tale sistema possa calcolare la similarità tra le parole e suggerire quelle più adatte ad un particolare contesto.*

Il meccanismo di suggerimento è indipendente dal contesto ed è in grado di funzionare anche per parole e espressioni che non siano necessariamente skills tecniche. Questo suo aspetto lo rende adatto ad essere impiegato anche in contesti reali, al di fuori quindi di quello universitario. La logica del sistema è separata dalla parte dell'interfaccia realizzata tramite i widgets di *Jupyter Notebook*, quindi, pensando ad un utilizzo professionale, questa può essere inserita all'interno di applicazioni web o di interfacce di programmi desktop.

## References

- [1] Giabelli, A., Malandri, L., Mercurio, F., Mezzanzanica, M. (2020). GraphLMI: A data driven system for exploring labor market information through graph databases. *Multimedia Tools and Applications*, 1-30.
- [2] Giabelli, A., Malandri, L., Mercurio, F., Mezzanzanica, M., & Seveso, A. (2020). NEO: A Tool for Taxonomy Enrichment with New Emerging Occupations.
- [3] CEDEFOP: Real-time labour market information on skill requirements: Setting up the eu system for online vacancy analysis. <https://goo.gl/5FZS3E> (2016).
- [4] Handbook, E. S. C. O. European Skills, Competences, Qualifications and Occupations (2017). EC Directorate E.