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 ultimi 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. Si rende necessario quindi individuare correttamente le competenze necessarie per svolgere un determinato lavoro. Assumono sempre maggiore importanza aree di studio come la Labor Market Intelligence (LMI), il cui obiettivo è di 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 report si descrive lo sviluppo di un sistema di suggerimento di skills definite dalla tassonomia standard europea delle occupazioni e delle competenze (E.S.C.O.). Il lavoro qui descritto sarà legato allo sviluppo di un sistema denominato skills4job, il quale richiede all'utente l'inserimento di skills di interesse e fornisce il nome di un lavoro ad esse correlato.

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 sintatticamente 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 contesto?*
In questo report si descrivono le fasi di sviluppo di un sistema di autocompletamento 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 partendo da studi precedenti per realizzare, in prima instanza 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 intorni a NEO sia molto ampio, in questa sede si descriveranno solo le 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 il processo che ha portato alla definizione della metrica utilizzata per fornire i suggerimenti. Nella sezione 3 si descrive l'architettura del sistema e i due moduli che la compongono. Nella sezione 4 si mostrano i risultati del test utente utile per valutare l'esperienza nell'utilizzo

del sistema. Nella sezione 5 si propongono alcuni scenari di studio da poter tenere in considerazione per futuri sviluppi del sistema e infine nella sezione 6 si presentano le conclusioni del progetto.

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 per l'arricchimento automatico della tassonomia E.S.C.O. (European Skills, Competences, Qualifications and Occupations) e i cui successivi sviluppi produrranno un sistema 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. La classificazione E.S.C.O. identifica e classifica skills, competenze, qualifiche e occupazioni rilevanti per il mercato del lavoro europeo, 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].
Il lavoro descritto nel paper vuole realizzare un sistema che (1) impari i word embeddings di concetti e entità delle tassonomia 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 si utilizzeranno i risultati ottenuti per il primo punto: 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, e 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 sotto-parole. Questo permette di gestire errori di typo nell'inserimento di parole e di 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 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 $C = \{c_1, \ldots, c_n\}$ delle skills già inserite dall'utente.

Per ogni input dell'utente si considera la stringa inserita fino a quel momento. Come misura di similarità si utilizza la similarità del coseno definita come:

$$\cos(\theta) = \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\| \cdot \|\vec{B}\|} \tag{1}$$

cioè il rapporto tra il prodotto scalare $\vec{A} \cdot \vec{B}$ di due vettori e il prodotto delle loro norme, i valori che può assumere sono compresi tra -1 e 1, dove -1 è perfettamente dissimile e 1 è perfettamente simile.
Si calcola 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:

$$A = \{(s, sim(i, s)) \mid s \in S \,\&\, s \notin C\} \tag{2}$$

Successivamente si considera il contesto e quindi, per ogni skill già inserita, si calcola la sua similarità con le skills non ancora inserite:

$$B = \{(s, sim_{avg}(s, C)) \mid s \in S \,\&\, s \notin C\} \tag{3}$$

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

$$sim_{avg}(s, C) = \frac{\sum_{c \in C} sim(s, c)}{|C|} \tag{4}$$

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 considerando sia l'input e sia 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 s si calcola:

$$sim_s = \alpha \cdot sim_A + (1 - \alpha) \cdot sim_B \qquad \begin{array}{l} \forall s \in S \,\&\, s \notin C \\ (s, sim_A) \in A \\ (s, sim_B) \in B \end{array} \tag{5}$$

Utilizzando l'equazione (5) si calcolano le similarità finali di tutte le skills e si suggerisce quella con valore di similarità massimo. L'utilizzo del parametro α permette al modello di essere flessibile dando maggiore importanza all'input corrente dell'utente o alle skills già inserite: per valori di α vicini a 1 aumenta l'importanza della skill che l'utente sta inserendo mentre per valori di α 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. In figura 3 è mostrato il workflow del sistema in cui si identificano due step: nel primo, ad ogni nuovo input dell'utente, vengono calcolate le similarità

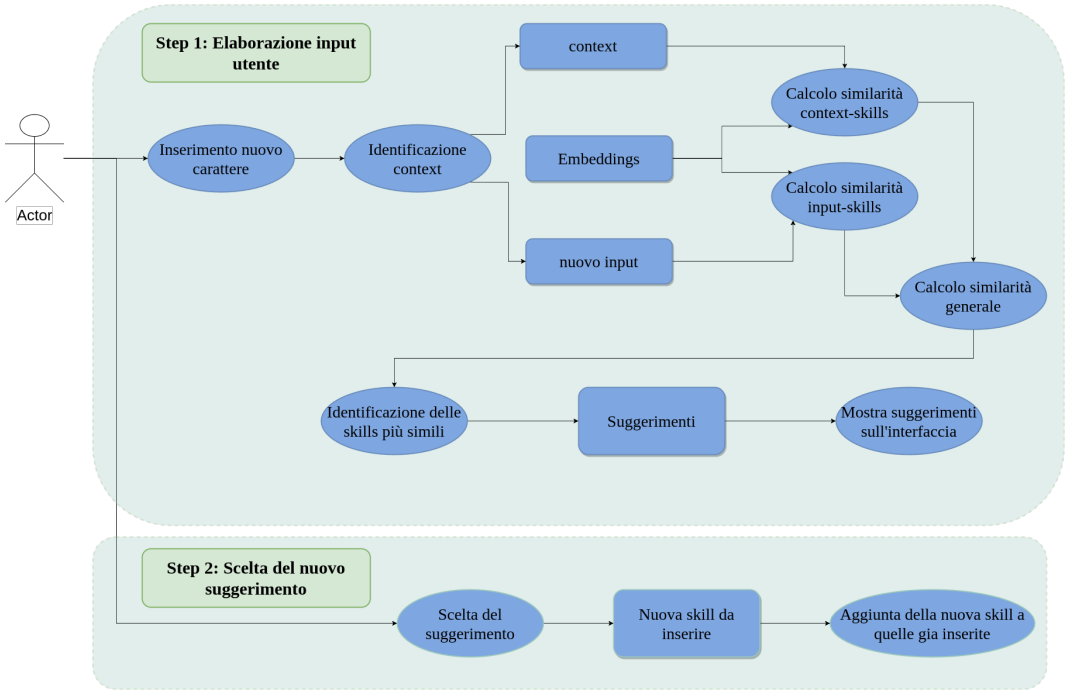

Figure 1: Workflow del sistema.

tramite gli embeddings, identificati i suggerimenti più appropriati e infine mostrati nell'interfaccia; nel secondo step il sistema è in attesa della scelta del suggerimento da parte dell'utente, una volta avvenuta viene aggiornata l'area di testo inserendo la skill scelta tra quelle già inserite.
L'implementazione è realizzata 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 2, 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.
Si è cercato di implementare il sistema in modo che sia più flessibile possibile e facilmente riutilizzabile in contesti diversi, questo si è tradotto nella scelta dei parametri di inizializzazione del sistema. Quando si effettua il deploy possono essere scelti alcuni parametri, come il numero di suggerimenti da mostrare (di default 4) e il valore di α usato per la pesatura delle similarità.

Fornendo una lista di parole di interesse e relativi embeddings, il sistema può essere utilizzato in diversi scenari.

3.1 Modulo AutoCompleteManager

Il modulo è stato realizzato come classe *Python* e ha il compito di gestire l'input dell'utente, calcolare le similarità e mostrare i suggerimenti. All'area di testo è associata una funzione *listener* che viene eseguita ad ogni nuovo carattere inserito, se l'inserimento di caratteri avviene velocemente si crea quindi una catena di esecuzioni del listener che rende l'esperienza utente difficoltosa rallentando il sistema. La soluzione proposta consiste nel rendere la funzione listener asincrona e ritardare la sua esecuzione fino a quando il valore non è stato modificato per un certo tempo, dopodiché il listener viene chiamato con il valore più recente. Questa strategia è stata implementata definendo la classe *Delay* che riceve in input il tempo di attesa e la funzione da eseguire al suo termine, la classe viene poi richiamata nel listener durante l'esecuzione di quest'ultimo.
Il modulo *AutoCompleteManager* si compone quindi di 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 α 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 (2). 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 invece l'insieme B definito nell'equazione (3). Utilizza anch'essa 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 (5) richiamando le due funzioni precedenti, se ne mostra la pseudo-codice:

Algorithm 1: *get_best_similarity_skill:* calcola le similarità secondo l'eq. (5)

Input: α, *new_input: l'input dell'utente, context: lista delle skills già inserite, num_suggests: numero di suggerimenti da mostrare, skills_list: le skills E.S.C.O.*

Output: Le skill più simili sia al contest (se presente) che all'input con i relativi valori di similarità

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

- suggest_interest_skill consiglia le skills maggiormente correlate al contesto ignorando l'input corrente dell'utente, richiama quindi la funzione get_skills_context_similarity.

- 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 debounce gestisce l'esecuzione ritardata del listener andando ad utilizzare la classe *Delay* come descritto precedentemente.

- La funzione suggests_manager è la funzione di listener associata all'area di testo nella 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 funzionie mostra i bottoni associati alle skills:

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

```
while True do
    new_input ← strings inserita fino a quel momento
    context ← lista delle skills già inserite
    suggests ← get_best_similarity_skill(new_input, context)
    for s in suggests do
        Crea e visualizza il bottone con descrizione s e la sua
         similarità con new_input
```

- 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 l'interfaccia del sistema ed è stato realizzato utilizzando i widgets di *Jupyter Notebook*. Come mostrato in figura 2, il form è composto da tre sezioni distinte: la

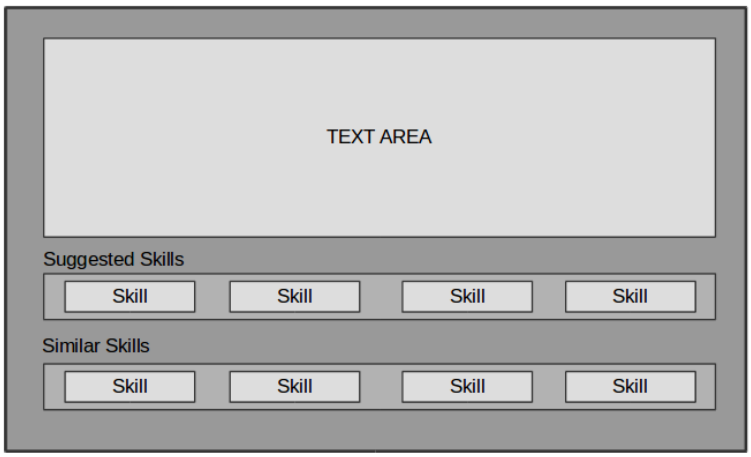

Figure 2: Struttura dell'interfaccia utente.

text area, con cui l'utente inserisce le skills di interesse e in cui si autocompleta l'input secondo il suggerimento scelto, la sezione *Suggested Skills* in cui sono presenti i bottoni etichettati con il nome della skill e la loro similarità; in questa sezione i suggerimenti sono calcolati considerando sia l'input che

il contesto tramite la loro combinazione lineare. La sezione *Similar Skills* mostra 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*, in seguito alla scelta dell'utente permettendo la visualizzazione di bottoni con nuovi suggerimenti.

Nella figura 3 si può vedere l'interfaccia utente in attività durante l'interazione, sono presenti le tre sezioni descritte con i relativi suggerimenti.

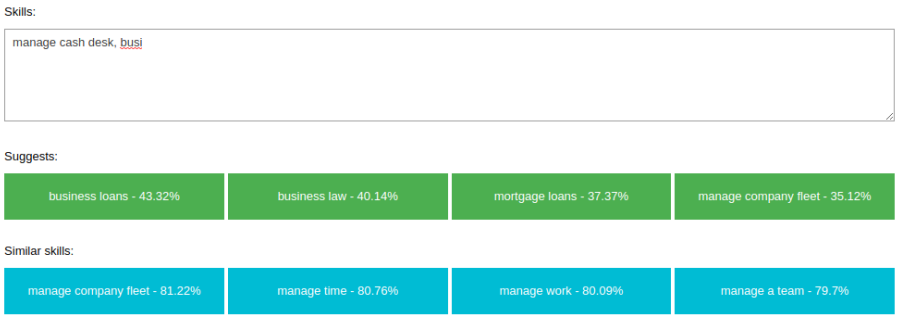

Figure 3: Interfaccia utente durante l'interazione.

4 User Test

Per valutare l'interazione degli utenti con il sistema è stato realizzato un user test tramite un questionario. Gli utenti, dopo aver utilizzato il sistema per almeno cinque minuti, hanno dovuto rispondere ad una serie di domande per valutare la loro esperienza. Sono state identificate delle caratteristiche da valutare ed è stata creata una scala di cinque valori per ognuna delle seguenti caratteristica:

- Velocità: valutare la velocità di risposta del sistema, da *lento* (valore 1) a *veloce* (valore 5).
- Originalità: valutare quanto il sistema sia originale rispetto ad altri sistemi simili, da *convenzionale* (valore 1) a *originale* (valore 5).
- Supporto: valutare quanto il sistema possa essere di supporto alle necessità dell'utente che lo utilizza, da *di ostacolo* (valore 1) a *di supporto* (valore 5).
- Facilità di utilizzo: da *complicato* (valore 1) a *facile* (valore 5).
- Chiarezza nell'interazione: valutare la chiarezza delle risposte del sistema, da *confuso* (valore 1) a *chiaro* (valore 5).
- Ordine: valutare la disposizione dei componenti dell'interfaccia, da *Sovraccarico* (valore 1) a *Ordinato* (valore 5).

Il questionario è stato sottoposto ad un totale di 18 utenti, la figura 4 mostra la distribuzione del sesso e dell'età nel campione. Si mostrano i risultati del

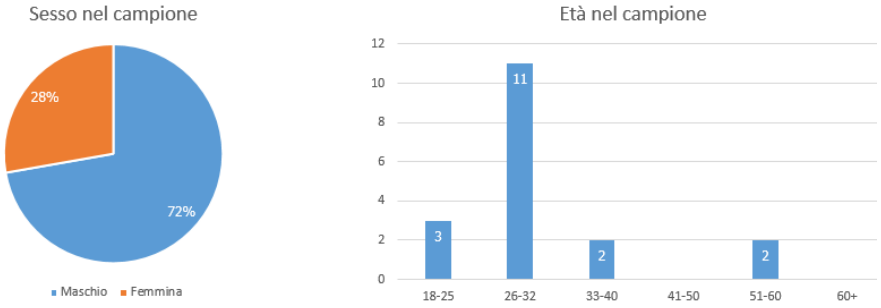

Figure 4: Composizione del campione.

test, nello stacked bar chart in figura 5 si vede, per ogni caratteristica da valutare, il numero delle risposte per ogni possibile valore. La maggior parte dei valori dati sono il *4* e il *5*, questo testimonia una buona risposta del sistema.

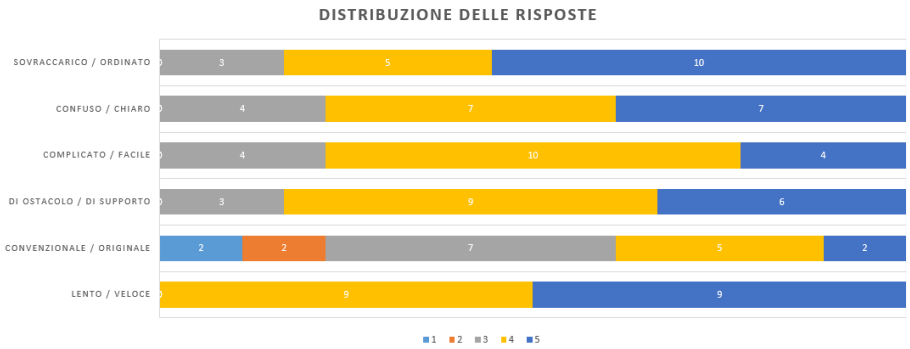

Figure 5: Distribuzione delle risposte.

In figura 6 sono mostrate le medie delle valutazioni per ogni caratteristica, si nota che solo la caratteristica *Convenzionale/originale* è significativamente inferiore rispetto alle altre medie.

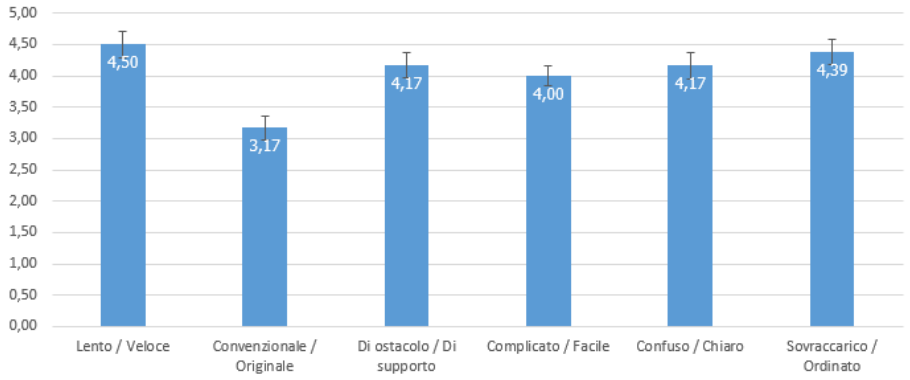

Figure 6: Medie dei valori per le varie caratteristiche con intervallo di confidenza al 95%.

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 (5) 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 ge-

ografica di pubblicazione, quindi ricalcolare gli embeddings considerando solo gli annunci filtrati per queste caratteristiche.
Ulteriore oggetto di studio può essere, infine, il comportamento del sistema se si utilizzassero 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 sintatticamente 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 (5) 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 dallo scenario di utilizzo ed è in grado di funzionare anche per parole ed 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, essendo separata dalla parte dell'interfaccia, è facilmente inseribile all'interno di applicazioni web o di interfacce di programmi desktop.

References

[1] Giabelli, A., Malandri, L., Mercorio, 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., Mercorio, 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.