Scuola universitaria professionale della Svizzera italiana
Dipartimento tecnologie innovative

SUPSI

Machine Learning for disambiguation of scientific article authors

Studente/i

Brian Pulfer

Relatore

Dr. Fabio Rinaldi

Correlatore

Prof. Luca Maria Gambardella

Committente

F. Hoffmann-La Roche AG

Corso di laurea

Bachelor of Science SUPSI in Ingegneria informatica

Modulo

M00009 - Progetto di Diploma

Anno

2019

Data

30 agosto 2019

Indice

Abstract

Ad oggi, vi è grande ambiguità nell'identificare univocamente l'autore di un articolo scientifico. Questo è dovuto da diversi fattori, come ad esempio dal fatto che sino a poco tempo fa' molti quotidiani erano propensi a pubblicare solo l'iniziale del nome dello scienziato a volte omettendo la sua affiliazione, informazione comunque difficile da tracciare. Un'altro fattore che genera ambiguità è la trascrizione di nomi asiatici in testo latino, cosa che comporta spesso e volentieri perdita di informazioni.

La soluzione a questa problematica è nota come AND (Author Name Disambiguation) ed è parzialmente realizzabile con delle tecniche di *machine learning*. La risoluzione del problema ha come scopo più grande quello di portare un maggiore vantaggio competitivo nel settore delle risorse umane delle società scientifiche.

Lo scopo del progetto è quello di sviluppare dei sistemi di *machine learning* capaci di distinguere a quali autori appartenga quale articolo scientifico e saper distinguerne il migliore. Gli algoritmi verranno applicati al database PubMed.

To date, there is big ambiguity in univocally identifying the author of a scientific article. This is caused by different factors, for example the fact that until recently many journals were inclined to only publish the initial in place of the first name of the scientist and sometimes omitting his affiliation, which is an already hard-to-track information. Another factor that generates ambiguity is the transcription of asian names into latin text, which often leads to information loss.

The solution to this problem is known as AND (Author Name Disambiguation) and it's partially realizable with some machine learning techniques. The problem's resolution has as greater goal bringing a bigger competitive vantage in the human resources sector of scien-

tific societies.

The goal of the project is to develop some machine learning systems capable of distinguish to which authors belongs which scientific article and picking the best. The algorithms will be applied to the PubMed database.

Progetto assegnato

Machine learning for disambiguation of scientific article authors

Descrizione:
Names of authors of scientific publications are very often underspecified leading to a high degree of ambiguity. For example, until recently it was customary in several journals to use only the initial in place of the first name. Affiliation is not always given, and in any case when authors change Institution it becomes difficult to track them, especially for very common names. Additionally, translitteration of Asian names into the latin script is another major source of ambiguity. The goal of the project is to devise efficient algorithms to cross reference authors from millions of publications, apply such algorithms to the PubMed database, and correlate authors with descriptors of their scientific output. This project builds upon previous work done in collaboration with a major Swiss Pharmaceutical company with the larger goal of providing a competitive advantage in the area of human resources.

Compiti:
- development of a machine learning system capable of disambiguating authors' names
- testing of the system across a large database of scientific publications
- addition of descriptors about scientific output of authors

Obiettivi:
- Implementation and testing of the above system

Tecnologie:
- Python
- Machine Learning toolboxes

Contatto esterno:
Nessun contatto esterno presente

Allegati: Nessun Allegato

Capitolo 1

Obiettivi

Lo scopo del progetto è quello di implementare, addestrare e testare, nel linguaggio di programmazione **python**, un classificatore di apprendimento automatico in grado di predire se una copia di articoli scientifici appartiene o meno allo stesso autore.

Questi articoli scientifici sono inerenti al campo della biomedicina e sono estratti da **PubMed**, un motore di ricerca gratuito che contiene oltre 29'000'000 di citazioni di articoli biomedici, vari journal e libri online e rappresenta la libreria medica nazionale statunitense.

Capitolo 2

Lavori correlati

2.1 Progetto precedente

Il progetto consiste in realtà di un implementazione in un diverso linguaggio di programmazione di un tool sviluppato già in precedenza nel linguaggio di programmazione Java.

Il lavoro svolto in precedenza[1] su questo progetto permette di affrontare da subito gli studi su un *gold standard database*, ovvero su quello di **PubMed**[2].

Figura 2.1: Logo PubMed

PubMed è un motore di ricerca gratuito che contiene oltre 29 milioni di citazioni di articoli biomedici presi dal database MEDLINE, vari *journal* e libri online e rappresenta la libreria medica nazionale statunitense.

Il database contiene però solo le citazioni e gli *abstract* dei **singoli** articoli scientifici, pertanto è stato necessario nel passato generare delle coppie di articoli il cui nome dell'autore era ambiguo. Per fare ciò si sono creati dei *namespaces*. Due articoli appartengono allo stesso *namespace* se il cognome degli autori è identico e se pure la sola iniziale dei loro nomi lo è.

A questo punto è possibile a partire da un *namespace* contenente N articoli generare (N-1)+(N-2)+(N-3)...+(N-(N-1)) coppie di articoli il cui autore è ambiguo.

Tramite crowdsourcing, pareri di esperti ed e-mail agli autori di queste pubblicazioni è stato poi possibile assegnare un *label* (classe: la coppia di articoli appartiene allo stesso autore / la coppia di articoli non appartiene allo stesso autore) ad alcune coppie di articoli.

Le coppie di articoli fornite di *label* rappresentano il database utilizzato per AND che è composto da 1'900 osservazioni.

In seguito si è diviso il nuovo *dataset* in due: una parte di *training set* contenente 1'500 istanze ed una parte di *testing set* contenente le rimanenti 400 osservazioni. Allenando classificatori di diverso tipo su questo dataset ed andando ad analizzare nuove *features* che si sono rivelate importanti (**journal descriptors, semantic types, ambiguity score, last name length**), si è riusciti ad ottenere un classificatore con una precisione approssimativa del 90% applicando una *10-fold cross validation*.

2.2 Altri progetti di AND

Il problema dell'ambiguità dei nomi colpisce non solo il campo della biomedicina, ma la totalità della comunicazione accademica ed è dunque stato studiato con diversi approcci. Altri progetti hanno utilizzato diverse tecniche per poter fare distinzione fra articoli, come ad esempio l'utilizzo di *word embeddings* come *features* per il classificatore[3], o come l'utilizzo di svariati algoritmi come SVM[4] o Random Forest[5] ed addirittura algoritmi di *machine learning* non supervisionato[6].

In questo progetto, le tecniche sopraccitate sono state applicate con l'intenzione di aumentare la precisione del classificatore.

Capitolo 3

Descrizione dei dati

Per poter addestrare un classificatore che possa distinguere se due articoli scientifici appartengono o meno allo stesso autore, è necessario che questo sia addestrato con delle coppie di articoli (osservazioni) e le relative informazioni che indicano quali coppie appartengono allo stesso autore e quali no (*labels*).

Ogni osservazione contiene delle *features*, e ciascuna di queste esprime un confronto fra i due articoli. Un'osservazione è quindi anche interpretabile come un vettore unidimensionale dove ogni elemento del vettore esprime in valore assoluto una analogia o una differenza fra gli articoli.

3.1 Informazioni disponibili

Il *dataset* presente per l'addestramento del classificatore è una coppia di file in formato csv (*'comma-separated values'*, ovvero 'valori separati da virgole') dove ogni riga rappresenta una coppia di articoli e presenta le seguenti informazioni:

PMID1	Cognome1	Iniziali1	Nome1	PMID2	Cognome2	Iniziali2	Nome2	Appartenenza

Tabella 3.1: Riga generica del dataset

Dove **PMID** (PubMed ID) è l'id dell'articolo. Le prime 4 colonne di ogni riga sono inerenti al primo articolo, mentre le seconde 4 colonne al secondo.
Mentre queste prime 8 colonne contengono sempre dati variabili, l'ultima colonna è formata sempre da un valore binario che sta' ad indicare se i due autori degli articoli siano in realtà la stessa persona oppure no.

È evidente che non ha nessun senso confrontare gli ID degli articoli per stabilirne la proprie-

tà, mentre sono più interessanti le informazioni relative a nome, cognome ed iniziali. Ciò nonostante, appare presto chiaro come queste informazioni non siano sufficienti.

3.1.1 Estrazione degli articoli

Per poter stabilire se a scrivere due articoli sia stato lo stesso autore, conviene estrarre più informazioni dagli articoli stessi.

Questo è possibile grazie alla *API (Application Programming Interface)* fornita da PubMed per la ricerca ed il download degli articoli presenti nel suo database, ovvero **e-utilities**.

E-utilities è un set di otto programmi *server-side* che forniscono un'interfaccia di accesso al database del centro nazionale per l'informazione sulla biotecnologia statunitense (NCBI) e dunque, anche al database di PubMed.

Tramite le E-Utilities è possibile, a partire dall'ID PubMed di un articolo, eseguirne il *fetch* (download) in formato XML e derivarne le informazioni. Le informazioni estratte da questi file riguardano indirizzo e-mail, data, locazione, termini MeSH (parole chiavi), nomi/cognomi/iniziali degli autori e co-autori, nome dell'affiliazione e lingua in cui è scritto l'articolo.

Per questioni di prestazioni e scalabilità, si è preferito scaricare localmente gli articoli che componevano il dataset ed eseguire letture da file anziché eseguire richieste di *fetch* per ogni articolo al momento dell'addestramento del classificatore. Dalle 1'900 coppie di articoli, sono emersi **3'734 articoli distinti**. Gli articoli sono stati scaricati in formato xml ed occupano uno spazio totale di all'incirca 60 Megabyte.

Seguendo la traccia del lavoro svolto in precedenza, si è deciso di utilizzare una libreria per il riconoscimento di **organizzazioni** e **luoghi** a partire da una stringa di testo[7]. Si è infatti sfruttata la nota libreria **Stanford NER** (Stanford Named Entity Recognizer) per utilizzare il suo modello 7-class addestrato sui dataset MUC 7. Le informazioni estratte con questa libreria sono state scritte su files per ogni articolo, e vengono caricate durante l'esecuzione del programma principale.

Nonostante l'utilizzo di informazioni di base riguardo all'articolo possa sembrare più che sufficiente, è in realtà di grande aiuto effettuare un'analisi testuale di entrambi gli articoli.

3.1.2 Analisi del testo via OGER

Figura 3.1: Logo OntoGene

OntoGene/**BioMeXT** è un'iniziativa di ricerca che punta ad ampliare i confini del *text mining* per la letteratura bio-medicale la cui competenza principale è quella dell'estrazione di informazione. L'iniziativa espone due risorse principali: **BTH** (Bio Term Hub) ed **OGER** (OntoGene Entity Recognition)[8].

Figura 3.2: Raffigurazione del funzionamento di BioTermHub

BTH è un aggregatore di terminologie biomedicali che permette la rapida costruzione di una risorsa di terminologie in un formato adatto al *text mining* (tsv). Le terminologie sono estratte da *databases* ben affermati in campo medico (es: ChEBI, NCBI, Cell Onthology, . . .).

OGER è una libreria open-source che utilizza un file costruito con BTH per annotare gli articoli scientifici.

La libreria **OGER** è utilizzata nel progetto per l'estrazione delle annotazioni significative. Il file delle annotazioni da estrarre (tsv) utilizzato è quello di default fornito con il download della libreria.

Pure in questo caso, si è preferito scrivere le informazioni in un file per ogni articolo per questioni di prestazioni e scalabilità. In ogni documento sono scritti gli ID delle entità rilevate nel testo dell'articolo.

3.1.3 Analisi del testo via Text Categorization 2011

Text Categorization è una libreria Java open-source, la cui ultima *release* risale al 2011, che permette l'estrazione di *Journal Descriptors* e *Semantic Types* da una stringa di testo[9].

I *Journal Descriptors* sono dei descrittori dell'articolo. Sono simili a delle parole chiavi, ma forniscono maggior dettaglio descrivendo le diverse specialità associate all'articolo ed individuando non solo il dominio principale ma anche quelli secondari. Esempi di *Journal Descriptors* sono: 'Pathology', 'Neoplasms' e 'Pulmonary Medicine'.

I *Semantic Types* rappresentano invece i diversi possibili tipi semantici. Esempi di *semantic types* sono: 'Pathologic Function', 'Spatial Concept' e 'Drug Delivery Device'.

Si noti che i *Journal Descriptors* ed i *Semantic Types*, come pure le *entities* estratte con la libreria OGER, non sono termini direttamente presenti nel testo, ma derivati tramite l'analisi testuale.

In totale esistono 122 descrittori e 135 tipi semantici.

Per poter incorporare queste *features*, che si sono rivelate quasi discriminatorie nel lavoro svolto in precedenza, è necessario quindi del codice Java che faccia utilizzo della libreria (chiamata **tc2011**) e che fornisca tali informazioni al codice python. La nota libreria **Pyjnius** permette proprio l'utilizzo di classi Java all'interno del codice python.

È stato dunque implementato un script python che grazie all'utilizzo di Pyjnius ha permesso l'esposizione di metodi necessari per l'estrazione delle *features* d'interesse.

Tali *features* sono state scritte su files per questioni di prestazioni. Questi files sono letti dal programma principale.

3.1.4 Feature inerente l'ambiguità dei nomi

Seguendo la traccia del lavoro svolto in precedenza, dove risultava che un'indicazione su quanto un nome fosse ambiguo aiutasse il classificatore, si è deciso pure in questo progetto di considerare una tale *feature*.

Questa *feature* nasce dal intuitivo concetto che se il cognome di un autore è molto popolare, allora risulterà più improbabile che due articoli aventi questo cognome popolare appartengano allo stesso autore. Di riflesso se un cognome è molto inusuale risulterà più difficile che sia condiviso da più autori.

Per il calcolo dell'ambiguità di un articolo, si considerano il cognome completo e le iniziali. Viene fatto un rapporto fra la dimensione del *namespace* e la dimensione dell'intero *dataset*.

3.1.5 Feature inerente la lunghezza del cognome

Sempre seguendo il lavoro svolto in precedenza e con la consapevolezza anticipata che una tale *feature* può aiutare un qualsiasi classificatore, si è deciso anche in questo progetto di allenare i classificatori con una indicazione sulla lunghezza del cognome ambiguo. Intuitivamente, un cognome lungo (di norma) è meno comune di un cognome corto e in quanto tale è meno probabile che appartenga a due autori diversi e vice-versa.

3.1.6 Vettore dell'articolo

Uno dei più recenti ed affascinanti metodi per confrontare degli articoli, o più in generale dei documenti, è il meccanismo offerto da delle reti neurali per assegnare ad un documento un vettore con un numero di dimensioni parametrizzabile. Tale tecnica è nota come **Doc2Vec** e deriva dalla meno recente tecnica **Word2Vec**, che permette l'assegnamento di un vettore ad una parola[10].

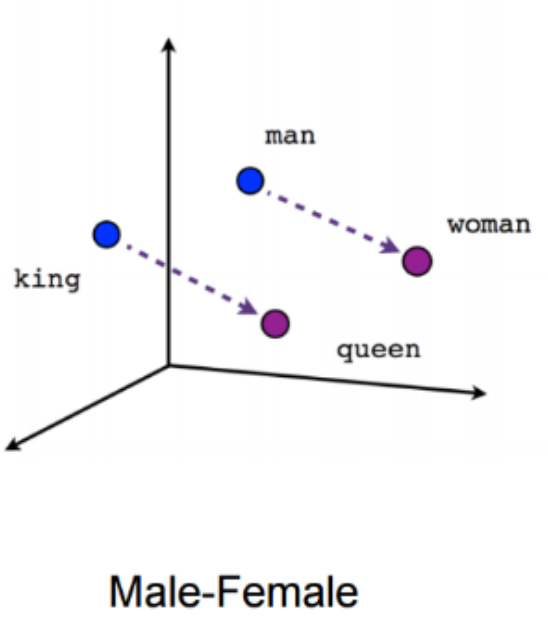

Figura 3.3: Esempio di interpretazione dei vettori generati da Word2Vec

L'esempio classico fornito per spiegare l'utilità di Word2Vec, raffigurato in Figura 3.3, è quello del Re e della Regina: una volta che si ottiene dalla rete neurale un vettore per le parole: 'Re', 'Regina', 'Uomo', 'Donna', è possibile accorgersi come la distanza fra i vettori 'Re' e 'Regina' sia la stessa (o molto simile) a quella fra 'Uomo' e 'Donna'. Applicando dell'algebra lineare inoltre, è possible sottrarre al vettore della parola 'Re' il vettore della parola 'Uomo' ed aggiungere il vettore della parola 'Donna' ottenendo così il vettore della parola 'Regina'.

Questo stesso concetto è applicabile pure ad interi documenti, dove ogni documento è collocato in uno spazio N-dimensionale. Nonostante in questo progetto non sia necessario effetture delle somme fra i vettori dei documenti, è utile associare a due documenti un vettore e calcolarne il coseno dell'angolo fra gli stessi. Una bassa differenza angolare fra i vettori degli articoli è sinonimo del fatto che gli articoli trattino una tematica simile e, considerando che uno stesso autore probabilmente scriverà articoli su temi quantomeno simili, una bassa differenza angolare fra i vettori degli articoli è sinonimo di una più alta possiblità che l'autore dei due articoli sia la stessa persona.

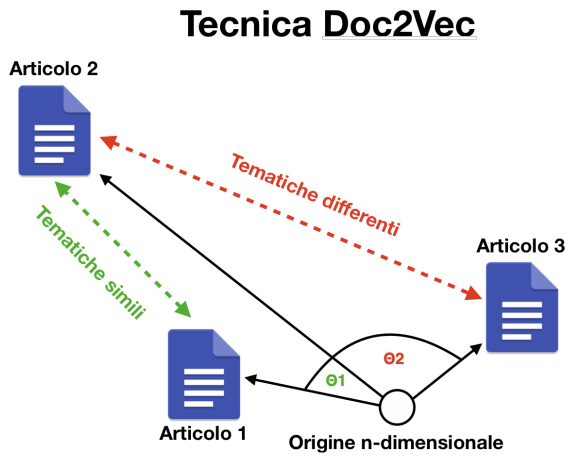

Figura 3.4: Esempio di interpretazione dei vettori generati da Doc2Vec

La rete neurale utilizzata è offerta dalla libreria python open-source **gensim**, ed è stata addestrata sulla totalità del dataset. Il layer nascosto, ovvero il layer che permette di stabilire la dimensione dei vettori, è composto da 300 nodi (quindi, 300 componenti per vettore).

Figura 3.5: Logo gensim

Una volta addestrato non è necessario testare il modello ma semplicemente recuperare le componenti dei vettori, che altro non sono che i valori dei pesi presenti sui nodi fra il layer di input ed il layer nascosto. Inoltre il modello non è addestrato ad ogni esecuzione del programma, ma una volta addestrato è persistito su file e recuperato per l'esecuzione del programma principale.

3.2 Confronto fra articoli

Il confronto fra articoli è necessario per poter creare le osservazioni di *training* e di *testing* del classificatore. Qui di seguito, vengono spiegate le metodologie adottate per tradurre le analogie e differenze fra gli articoli in valori numerici.

3.2.1 Nomi

Per confrontare i nomi degli autori principali degli articoli si fa' uso della **distanza di Levenshtein**[11].

La distanza di Levenshtein è una funzione di distanza fra due stringhe di testo. Date due stringhe in input, la distanza di Levenshtein ritorna il numero minimo di inserzioni, rimozioni e sostituzioni di lettere necessarie perché le due stringhe diventino identiche. Ad esempio, la distanza di Levenshtein fra '**toro**' ed '**oro**', fra '**toro**' e '**foro**' e fra '**toro**' e '**torto**' è 1.

Utilizzare la distanza di Levenshtein per confrontare la somiglianza fra i nomi degli autori permette di addestrare il classificatore con una maggiore precisione in quanto l'alternativa è identificare semplicemente se le stringhe di testo sono identiche o meno.

Minore sarà dunque la distanza di Levenshtein fra due nomi, maggiore sarà la possibilità che gli articoli appartengano allo stesso autore e vice versa.

3.2.2 Date

Tramite la libreria *built-in* di python **datetime** è possibile calcolare una differenza in anni fra due date, che è il valore assunto dalla *feature*.

3.2.3 Indirizzi e-mail

Quella per l'indirizzo e-mail è una feature binaria: vale ('1') se gli articoli condividono lo stesso indirizzo e-mail oppure ('0') nel caso opposto. Non si considera la differenza fra lettere maiuscole e minuscole.

3.2.4 Termini MeSH

Il valore inserito per questa *feature* sono il numero di termini MeSH (parole chiavi) identiche condivise da entrambi gli articoli.

3.2.5 Co-autori

Come per le parole chiave, il valore inserito per questa *feature* sono il numero di co-autori in comune fra gli articoli. Due articoli condividono lo stesso autore se ne condividono sia il nome che il cognome e le iniziali. Non si considerano le differenze fra lettere maiuscole e minuscole.

3.2.6 Paesi e città

Queste informazioni sono estratte dalla stringa riguardante l'affiliazione con la libreria **Stanford NER**. Si effettua un conteggio delle parole della locazione condivise fra gli articoli. Ad esempio: 'Milano, Italy' e 'Napoli, Italy' condividono una parola, ed il conteggio fra questi due articoli sarebbe dunque 1.

3.2.7 Affiliazione

La stringa dell'affiliazione è estratta con la libreria Stanford NER ed è poi divisa in una lista di parole. Per confrontare le affiliazioni, si calcola il rapporto fra il numero di parole condivise ed il numero medio di parole della stringa dell'organizzazione. Ad esempio, un confronto per le affiliazioni 'Scuola Universitaria Professionale della Svizzera Italiana' e 'Università della Svizzera Italiana' risulta in un punteggio di 3/5.

3.2.8 Tipo e descrittore dell'affiliazione

Come per la *feature* inerente la località (paesi e città) e l'affiliazione, l'informazione relativa all'affiliazione è estratta grazie alla libreria Stanford NER ed è scritta su file per ogni articolo. Si effettua un controllo per verificare se vi sia un *match* fra i tipi di organizzazioni, i descrittori, nessuno o entrambi.

Nella seguenti tabelle, sono rappresentati ogni **tipo** e **descrittore** di organizzazione considerati:

Tipo
UNIVERS
INSTITU
COLLEGE
LABOR
ORGANI
MINISTRY
CENTER
DEPARTMENT
HOSPITAL
SCHOOL

Tabella 3.2: Tipi di organizzazione

Descrittore
BIOLOG
CHEMIST
PEDIATRIC
SURGERY
MEDIC
GENETIC
INFECT
AGRICULT
ENTOMOLOG
BIOTECH
NEUROLOG
PSYCHOL
PHARMA
TOXIC
CANCER
CARDIOL
DENTIST
NUTRITION
HEALTH
DISEASE

Tabella 3.3: Descrittori di organizzazione

Viene inserito il valore ('0') per le coppie di articoli le cui affiliazioni non condividono né tipo né descrittore, ('1') se condividono uno dei due, ('2') se sia tipo che descrittore sono identici.

3.2.9 Tipi semantici e descrittori

Il valore per questa featuree è il numero di tipi semantici e descrittori in comune fra i due articoli. I descrittori ed i tipi semantici sono raffigurati come stringhe di testo.

3.2.10 Annotazioni

Per le annotazioni estratte con OGER, il valore inserito nell'osservazione (coppia di articoli) sarà il numero di annotazioni che entrambi gli articoli condividono. Si confrontano infatti gli ID delle entità di ciascun articolo.

3.2.11 Livelli di ambiguità del namespace

Il punteggio fornito al classificatore è il rapporto fra la dimensione del namespace e la dimensione del intero dataset.

3.2.12 Lunghezza del cognome

I cognomi sono identici per tutte le coppie di articoli del dataset, è quindi sufficiente fornire al classificatore la lunghezza del cognome del autore di uno dei due articoli.

3.2.13 Angolo fra i vettori degli articoli

Al classificatore è fornita per ogni coppia di articoli il coseno dell'angolo fra i relativi vettori (300 componenti ciascuno).

3.2.14 Lingue

Feature binaria che indica se gli articoli condividono la stessa lingua ('1') oppure no ('0').

3.2.15 Iniziali degli autori principali

Feature binaria che indica se gli articoli condividono le stesse iniziali per l'autore principale ('1') oppure no ('0').

3.3 Features mancanti

Con l'estrazione di tutte le *features*, emerge come molti dati di quelli estratti siano assenti per svariate coppie di articoli. Infatti, delle 1'500 coppie di articoli di training e 400 di testing (in totale solo 1889 a causa dell'assenza del ID di alcuni articoli) solo 144 e 36 sono complete di tutte le informazioni necessarie per costruire le *features* (e-mail, autori, data e termini mesh). Qui di seguito, una tabella che indica quante coppie di articoli sono assenti delle relative *features*:

Autori	Data	Termini MeSH	Indirizzo e-mail
10	10	269	1670

Tabella 3.4: Features mancanti

In questi casi solitamente si assegna alle *features* mancanti dei valori coerenti con i valori delle altre *features* facendo utilizzo di regressione. Alternativamente è possibile inserire il **valore medio** per quella *feature* di tutto il *dataset* oppure inserire dei valori casuali. Gli ultimi due approcci sono quelli seguiti per questo progetto.

Si noti che i tipi semantici, descrittori ed altre *features* citate in precedenza non sono inserite nella tabella in quanto vengono estratte dal abstract e titolo dell'articolo, che sono sempre presenti. Non è dunque un'errore o un ostacolo il fatto che un articolo possa essere sprovvisto di questi termini.

L'assenza di così tante *features*, ed il dover dunque compensare quest'ultime con dei valori assegnati in maniera arbitraria, genera rumore nel *dataset* ed in qualsivoglia classificatore.

3.4 Normalizzazione dei dati

Per evitare che le *features* che assumono valori più alti abbiano peso maggiore nella scelta del classificatore è necessario ricorrere ad una standardizzazione.

Ad ogni *feature* è sottratto il suo valore medio, e questo valore è infine diviso per la deviazione standard di quella *feature*. Questa standardizzazione è anche conosciuta come **z-score normalization**.

Applicando questa standardizzazione non si tiene conto del semplice valore numerico della *feature*, ma del suo scostamento rispetto agli altri dati.

La normalizzazione non è applicata ai dati che assumono esclusivamente valori binari (informazioni 'stesse iniziali', 'stesso indirizzo e-mail' e 'stessa lingua').

Di seguito, una tabella riassuntiva della totalità delle *features* disponibili e della loro presenza.

Nome	Confronto	Presenza
Indirizzo e-mail	’1’ se identici, ’0’ se diversi	13%
Termini MeSH	Conteggio delle parole chiavi in comune	86%
Data	Distanza in anni	99%
Co-autori	Conteggio dei co-autori condivisi	99%
Lunghezza cognome	Il numero di lettere presenti nel cognome	99%
Iniziali	’1’ se stesse iniziali, ’0’ se diverse	99%
Lingua	’1’ se stessa lingua, ’0’ altrimenti	100%
Nome	Distanza di Levenshtein	100%
Locazione	Conteggio di parole in comune	100%
Organizzazione	% di parole condivise nel nome	100%
Tipo e descrittore	’0’, ’1’ o ’2’ a dipendenza del *matching*	100%
Entities OGER	Conteggio delle *entities* condivise dagli articoli	100%
STs e JDs	Conteggio dei tipi semantici e descrittori condivisi	100%
Punteggio di ambiguità	Dimensioni namespace/dataset	100%
Angolo vettoriale	Coseno dell’angolo fra i vettori a 300 dimensioni	100%

Tabella 3.5: Tabella riassuntiva delle features

Capitolo 4

Metodi

4.1 Versioni baseline

Per 'versione *baseline*' si intende una versione del classificatore dall'implementazione molto semplice ed, appunto, dai risultati basilari.

4.1.1 Classe maggioritaria

Una possibilità è l'utilizzo di un classificatore che classifica le osservazioni del *testing set* in maniera casuale. Trattandosi questo di un problema di classificazione binario, ovvero di un problema dove le possibili classi assegnabili sono solo due (stesso autore / autori diversi), su un set di dati dalle classi equamente distribuite tale classificatore otterrebbe mediamente una precisione approssimativa del 50%.

Nonostante questo sia un approccio valido, si è preferito invece utilizzare un classificatore che sfrutta il *training set* per riconoscere la classe più presente al suo interno. Al momento della predizione di un'osservazione, il classificatore classifica una qualsiasi osservazione come quella che ha appreso essere la più 'popolare' (frequente nel *training set*).

Avendo nel *training set* 1'500 osservazioni, delle quali 987 etichettate come 'appartenente allo stesso autore', questo classificatore classifica tutte le osservazioni del *testing set* come tali. Essendo quindi il *testing set* composto da 400 osservazioni, delle quali 217 effettivamente appartenenti alla categoria 'appartenente allo stesso autore', questo classificatore di base ottiene una precisione del 217 / 400 = **54.2%**.

4.1.2 Senza features di text mining

Una più raffinata versione *baseline* è una versione del classificatore che non fa' utilizzo di *features* estratte con tecniche di *text mining*, ovvero: descrittori, tipi semantici, entità e vettori dei documenti.

Le uniche *features* utilizzate dunque sono quelle presenti nei file xml degli articoli, ovvero le più intuitive in assoluto.

Il classificatore utilizzato è un classico classificatore utilizzato per AND ed è il classificatore **Random Forest**. Tale classificatore ottiene una precisione approssimativa sul *testing set* del **77.4%**, mentre con una validazione *10-fold* se ne ottiene una del **77%**.

Questa percentuale rappresenta il minimo raggiungibile per un classificatore finale.

4.2 Versioni di sviluppo

Le versioni di sviluppo del classificatore hanno come scopo principale il miglioramento della precisione sullo stesso set di dati. Per ottenere una precisione migliore di quella ottenuta con il classificatore *baseline* è possibile utilizzare più *features* (significative) oppure un diverso e miglior algoritmo di apprendimento/predizione.

Tutte le versioni sono state testate prima con l'utilizzo delle *features* di base e delle entities OGER. In seguito sono stati testati con la totalità delle *features* attualmente disponibili (descrittori, tipi semantici, punteggio di ambiguità e lunghezza del cognome).

4.2.1 K Nearest Neighbours (K-NN)

La prima versione implementata è un semplice classificatore **K-NN** (*K Nearest Neighbours*, ovvero 'i K vicini più prossimi') addestrato unicamente sulle *features* di base ed un unica *feature* di analisi testuale (entities fornite da OGER). Queste *features* sono i confronti fra: date, autori, parole chiavi, nomi delle affiliazioni, indirizzi e-mail, paesi, città, entities OGER.

In questo classificatore, le istanze di test sono classificate in base alla loro vicinanza (tipicamente distanza euclidea) con le K osservazioni più vicine. Alle istanze è data la stessa classe della maggior parte delle loro osservazioni più vicine. Il valore di K è definito a priori. Utilizzando la libreria **scikit learn** è stato possibile creare ed impiegare questo classificatore con diversi valori di K.

Scikit learn è una libreria python open-source di apprendimento automatico di semplice

utilizzo contenente una moltitudine di algoritmi di classificazione, regressione ed altri. Fra questi algoritmi vi è pure, appunto, l'algoritmo K-NN.

Figura 4.1: Logo di Scikit Learn

Il risultato migliore si ha per un valore di **K** uguale a 7. La precisione di tale classificatore ammonta al 74%. Questa percentuale è inferiore a quella del classificatore *baseline*, e non è dunque accettabile.

A seguito del successo nell'estrazione delle altre *features* significative (ambiguità dei nomi degli autori, *journal descriptors, semantic types*, lunghezza del cognome, vettori relativi agli articoli), applicando queste *features* la precisione per questo classificatore sale al **78%** circa.

Questo dato è sintomo dell'importanza delle *features* aggiunte nel secondo momento e rappresenta un sensibile miglioramento rispetto al classificatore di base.

4.2.2 Rete neurale feed-forward

Durante lo sviluppo del progetto si è implementato un classificatore di *machine learning* non supervisionato[6], in particolare una rete neurale con l'utilizzo della libreria **tensorflow**.

TensorFlow è una libreria open source che aiuta gli sviluppatori nell'implementazione ed addestramento di modelli di *machine learning* per i linguaggi python, javascript e swift.

Keras è l'API di alto livello per la creazione ed addestramento di modelli. Grazie a questa API, è possibile parametrizzare la rete neurale (ottimizzatore, *layers, loss function*, ...) che costituirà il classificatore.

La rete neurale impiegata è composta da 3 layers: un layer di input con 8 nodi, un layer nascosto con 5 nodi ed un layer di output con 2 nodi. Tutti i nodi del layer di input sono

Figura 4.2: Logo di TensorFlow

collegati con tutti i nodi del layer nascosto, e tutti i nodi del layer nascosto sono collegati con tutti i nodi del layer di output.

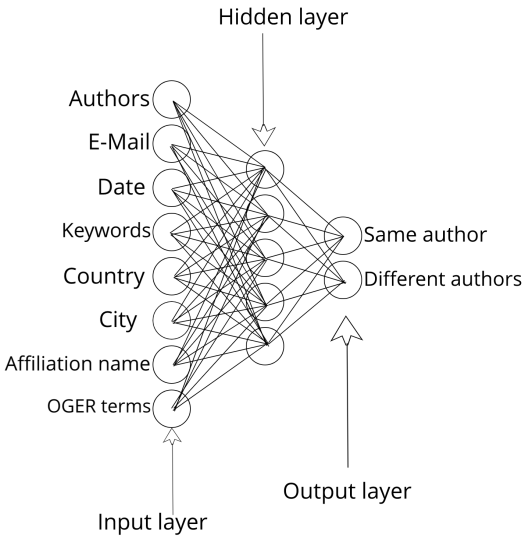

Figura 4.3: Rete neurale feed-forward

L'ottimizzatore utilizzato è un'istanza della classe *GradientDescentOptimizer* (utilizza l'algoritmo di gradient descent), mentre la funzione di *loss* è semplicemente la media degli errori al quadrato. La funzione di attivazione per i primi due layer è la funzione **ReLU**, mentre la funzione di attivazione per l'ultimo layer è la funzione **softmax**.

Cominciando con pesi e bias casuali, la precisione della rete è variabile. Dopo un numero soddisfacente di esecuzioni, si nota come la rete oscilli in precisione fra il 40% ed il 60%. I risultati per questo classificatore sono sicuramente insoddisfacenti.

Si noti come nella figura 4.3 manchino alcune *features* come tipi semantici e descrittori. Queste *features* sono assenti siccome al momento dell'implementazione della rete neurale ancora non erano disponibili.

Dopo l'aggiunta di queste *features*, la precisione della rete neurale non varia in maniera significativa. Questo è dovuto al fatto che i dati a disposizione sono troppo pochi, ed è dunque impossibile allenare la rete in maniera soddisfacente.

L'utilizzo di una rete neurale come classificatore finale è stato dunque escluso.

4.2.3 CART

Classification And Regression Trees è un algoritmo che permette di costruire, a partire da un *training set*, un albero decisionale per problemi di classificazione o di regressione.

Un albero decisionale altro non è che un albero che può essere percorso dalla radice verso il basso rispondendo alle domande riguardanti le *features* dell'osservazione presenti sui nodi. Ogni differente risposta porta ad un differente spostamento nell'albero. I nodi foglia rappresentano le classificazioni e sono il risultato stabilito dal classificatore in base alle risposte "date".

L'albero è costruito analizzando ogni *feature* presente in ogni osservazione e ponendo le domande più discriminanti come prime (più 'in alto').

La precisione per il classificatore senza l'utilizzo di descrittori, tipi semantici, lunghezza ed ambiguità del cognome e vettori associati agli articoli è del 71%. Dopo l'aggiunta di tali *features*, la precisione aumenta sino al **76%**.

4.2.4 Random forest

L'algoritmo '*Random Forest*', ovvero 'Foresta Casuale', prevede la costruzione di una moltitudine di alberi decisionali secondo diversi algoritmi, quali **ID3, C4.5, CART** ed altri[12].

La predizione della foresta sarà la predizione maggioritaria di tutti gli alberi decisionali.

Come per i classificatori K-NN e CART, questo classificatore è offerto dalla libreria **scikit learn** ed utilizza 50 alberi di decisione (siccome non sono visibili miglioramenti al di sopra di tale soglia).

Il risultato del classificatore senza l'utilizzo delle entità, dei descrittori, dei tipi semantici e dei vettori degli articoli si aggirava attorno al 77%. La percentuale coincide con il classificatore di base siccome questa versione è l'implementazione dello stesso.

Andando invece ad aggiungere le informazioni relative a descrittori, tipi semantici, il livello di ambiguità, la lunghezza del cognome e l'angolo compreso fra i vettori degli articoli la precisione del classificatore sale approssimativamente a **85**% sul *testing set*.

Avendo l'algoritmo stesso in sé una intrinseca componente casuale, i risultati variano sensibilmente di esecuzione in esecuzione.

4.2.5 SVM

L'algoritmo *Support Vector Machine* è un'algoritmo utilizzabile sia in problemi di classificazione che di regressione. L'obiettivo è quello di creare un hyper-piano che separi le istanze di una classe dall'altra attraverso l'utilizzo di vettori di supporto[13].

Pure questa implementazione è offerta dalla libreria scikit learn.

Utilizzando unicamente le *features* di base (nome, indirizzo e-mail, data...) il classificatore ottiene una precisione del 75%.

Una volta aggiunte le *features* relative a descrittori, tipi semantici, vettori degli articoli, lunghezza del cognome e relativa ambiguità, il classificatore ottiene una precisione sul testing set del **80%**.

4.3 Versione finale e Risultati

A seguito dei vari tentativi di sviluppo di classificatori, quello con la maggiore precisione in assoluto è il classificatore **Random Forest** con l'utilizzo di 50 o più alberi decisionali. Questo classificatore infatti offre una precisione approssimativa del **85%** sul testing set, mentre con l'utilizzo di una convalida incrociata a 10 parti (*10-fold cross validation*) si scopre una precisione del **87%**.

La **matrice di confusione** è una matrice costruibile nei problemi di classificazione binaria. Indica in che modo il classificatore indovina o sbaglia la classificazione.

	Classificazione Negativa	**Classificazione Positiva**
Osservazione Negativa	33.3%	12.1%
Osservazione Positiva	3.5%	51.1%

Tabella 4.1: Matrice di confusione del classificatore finale

Da questa matrice (Tabella 4.1), si osserva come il classificatore sia più efficace sulle osservazioni positive. Infatti, circa il 54.7% del testing set è composto da queste istanze ed il classificatore riesce a classificarne correttamente circa il 51.1%. Se valutiamo la precisione del classificatore sulle sole osservazioni positive, il classificatore ottiene una precisione approssimativa del 51.1 / 54.6 = **93.5%**.

Si nota anche come faccia più fatica invece quando l'osservazione da classificare sia negativa. Infatti circa il 45.3% del *testing set* è composto da osservazioni di classe negativa, ma il classificatore riesce a classificarne correttamente solo il 33.3%. Valutando unicamente la precisione sulle osservazioni negative, il classificatore ottiene una precisione approssimativa del 33.3 / 45.4 = **73.3%**.

Questa difficoltà può essere dovuta allo squilibrio fra osservazioni nel *training set*.

Un aspetto favorevole è il fatto che il classificatore sia più propenso a sbagliare classificando come 'appartenenti allo stesso autore' due articoli che in realtà non lo sono anziché classificando come 'non appartenenti allo stesso autore' due articoli che in realtà lo sono.

Questo è un aspetto molto favorevole in quanto lo scopo ultimo del software è quello di portare un vantaggio competitivo nel settore delle risorse umane di una realtà aziendale. I responsabili delle risorse umane finiscono per mettersi in contatto con i presunti autori degli articoli ed una volta in contatto con questi autori, è possibile scoprire realmente se questi siano o meno gli autori degli articoli. Se il rapporto di falsi negativi fosse maggiore, queste aziende finirebbero per perdere potenziale personale altamente qualificato.

Di seguito, una tabella che riassume l'importanza di ogni feature sul classificatore finale valutate su una singola esecuzione:

Feature	Rilevanza
Descrittori e Tipi semantici	21.1%
Nome	14.3%
Lunghezza del cognome	9.8%
Data	9.7%
Vettore Doc2Vec	7.4%
Co-autori	7.4%
Organizzazione	6.3%
Città e Paese	6.1%
Termini MeSH	5.2%
Entità OGER	5.2%
Livello di ambiguità	3.7%
Tipo e descrittore di organizzazione	2.1%
Lingua	0.7%
Indirizzo e-mail	0.6%
Iniziali	0.4%

Tabella 4.2: Importanza delle features

Dalla tabella emerge come la feature inerente i descrittori ed i tipi semantici sia la più incisiva in assoluto, seguita da quella riguardante il nome e la lunghezza del cognome. Risulta inoltre frustrante lo scarsissimo impatto della feature inerente l'indirizzo e-mail, giustificato dal fatto di essere assente nella stragrande maggioranza delle coppie di articoli (1670/1889 coppie).

Infine, può essere interessante analizzare il comportamento del classificatore finale tenendo in considerazione unicamente le coppie di articoli fornite di tutte le *features*, ovvero le coppie che non sono state 'sporcate' con dei valori medi o casuali. Si ricorda che il numero di tali osservazioni ammonta a 180, suddivise in 144 di *training* e 36 di *testing*.

Nonostante i dati a disposizione per il classificatore siano pochi, come poche siano le osservazioni di *testing* per poter affermare con sicurezza l'efficacia del classificatore, questo ottiene una precisione approssimativa del **94%** sul testing set è del **91%** con una validazione incrociata *10-fold*.

Questo netto miglioramento nella precisione del classificatore è simbolico del rumore che si viene a generare utilizzando i valori medi in sostituzione delle *features* non presenti e dell'importanza di queste ultime.

Capitolo 5

Test

Il *testing* costituisce un'elemento essenziale di ogni progetto informatico. Ne offre garanzia di qualità e ne favorisce il riutilizzo ed estensione.

Il codice prodotto è testato con il tool di testing *built-in* di python **unittest** e copre la totalità dei moduli e delle classi.

Capitolo 6

Sviluppi futuri e conclusioni

Il modello creato è sicuramente migliorabile. La scoperta ed aggiunta di nuove *features* rilevanti, il miglioramento dei confronti fra le informazioni degli articoli già presenti ed una migliore estrazione dell'informazione costituiscono i tre pilastri di sviluppo principali.

6.1 Miglioramento delle performances

Allo stato attuale, per caricare correttamente le informazioni che costituiscono una coppia di articoli è necessario effettuare ben 12 letture da file: 6 letture per ogni articolo per estrarre informazioni inerenti il testo, descrittori, tipi semantici, annotazioni OGER, locazione, tipo di organizzazione. Questo porta ad un tempo di esecuzione dell'intero programma che supera i 10 minuti.

Questo design è sicuramente sfavorevole, siccome si moltiplica un overhead dovuto alla lettura da uno storage più lento della memoria centrale.

Il problema è semplicemente risolvibile spostando le informazioni di ogn'uno dei 6 files in uno unico per ogni articolo. In questo modo, l'overhead da lettura da file sarebbe sei volte inferiore. Alternativamente, è possibile parallelizzare le letture tramite l'uso di *threads* per diminuire il tempo di esecuzione del programma.

6.2 Miglior confronto fra località

Allo stato attuale, il valore inserito per la *feature* inerente la località è il numero di parole condivise per ciascuna località individuata dalla libreria Stanford NER.

Un approccio migliore potrebbe essere calcolare la distanza in kilometri fra i luoghi di pubblicazione, dove una bassa distanza fra le località degli articoli significherebbe una maggiore probabilità che l'autore degli articoli sia la stessa persona e vice-versa.

6.3 Sfruttamento della 'proprietà transitiva'

Un approccio meno tradizionale, ma comunque molto promettente per il progetto, è quello invece di sfruttare la 'proprietà transitiva', ovvero basare le nuove classificazioni sulle classificazioni precedenti. Esempio: consideriamo gli articoli A, B e C. Supponiamo che le coppie di articoli (A,B) ed (A,C) siano classificate come appartenenti allo stesso autore, mentre la coppia (B,C) sia classificata come non appartenente allo stesso autore. In questo caso, si è dinanzi ad un evidente contradizione e gli approcci per risolverla sono molteplici. Si può infatti pensare di classificare tutte le coppie di articoli come la classe che sembra essere la più probabile.

Inoltre, questa proprietà potrebbe essere sfruttata per creare delle nuove osservazioni ed **ampliare il dataset**. Esempio: si considerino gli articoli X, Y e Z. Dal *training set* si evince che gli articoli (X,Y) appartengono allo stesso autore mentre gli articoli (X,Z) no. È possibile a questo punto creare una terza coppia di articoli (Y,Z) il cui *label* sarà negativo ('non appartenenti allo stesso autore') e sfruttare questa nuova coppia per allenare o testare il classificatore. L'unico caso in cui non è possibile generare nuove osservazioni è il caso in cui il *namespace* contenga solo osservazioni di classe negativa.

Riuscire ad ampliare, anche se in una misura limitata, il dataset potrebbe risultare in un miglioramento della precisione del classificatore finale.

Nel *dataset* per AND sono presenti 64 *namespaces* con più di una coppia di articoli, 8 dei quali composti da 3 coppie mentre i restanti 56 da 2.

6.4 Namespaces di PubMed

I ragionamenti sulla proprietà transitiva hanno portato ad uno studio del *database* PubMed più approfondito il quale obiettivo era quello di determinare quanti *namespaces* esistessero e che dimensioni avessero questi ultimi.

Per fare ciò, è stato sufficiente scaricare il database e posizionare virtualmente ogni autore di ogni articolo ad un *namespace* aumentandone il conteggio. Il risultato ultimo è un file in formato CSV contenente una colonna per il nome del *namespace* (ad. es. (Pulfer, B)) ed una colonna per il numero di autori contati nel relativo *namespace*.

Gli autori le cui informazioni sui nomi erano assenti sono stati esclusi.

Il codice per il download e decompressione del database come per quello di scrittura dei *namespaces* su file è stato scritto in python.

Il database (versione dicembre 2018) compresso ha una dimensione 27.5 gigabyte, mentre decompresso la dimensione totale si aggira attorno ai **235.1 gigabyte**.

Dal file generato si evince che nei oltre 29 milioni di articoli contenuti nel *database* PubMed sono contenuti **5'934'442** *namespaces* per un totale di 113'452'566 referenze ad autori (poco meno di 4 autori per articolo in media). Il *namespace* più numeroso in assoluto è **Wang Y** con ben **126'842** referenze, seguito da **Zhang Y** (108'513) e **Wang J** (101'007).

In media, ogni *namespace* contiene circa 19 elementi e la deviazione standard è di circa 256. La mediana assume il valore 2, menter la moda è 1, con ben 2'339'401 di *namespaces* che condividono questa cardinalità. Quest'ultimo dato inoltre ci informa sul fatto che i restanti *namespaces* contengono più di un'osservazione e possono dunque risultare ambigui.

Nel seguente *barchart*, viene raffigurato il numero di namespaces avente come cardinalità i rispettivi valori.

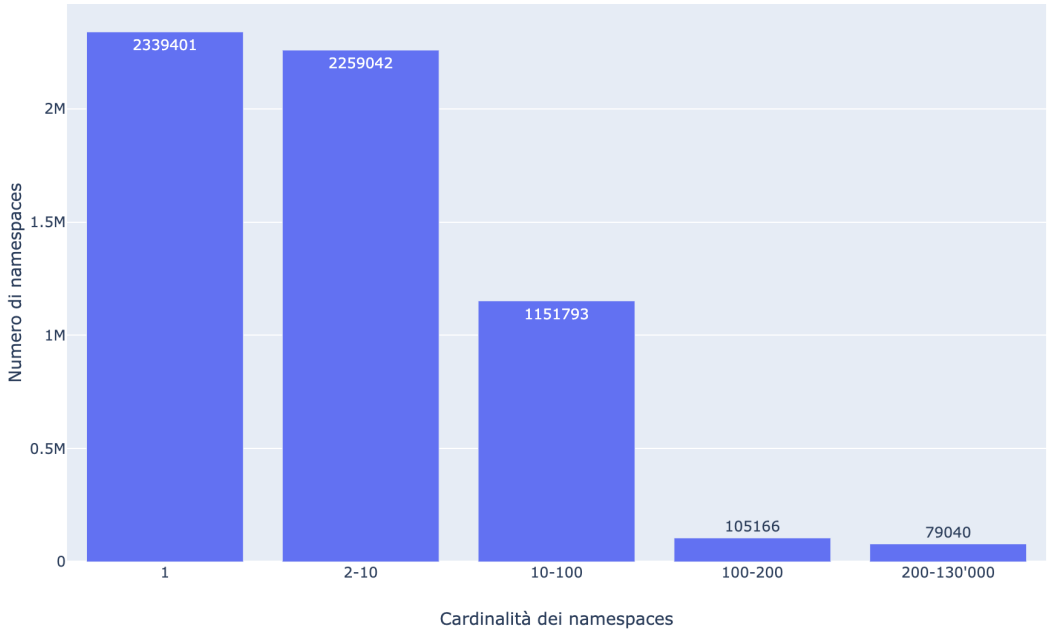

Figura 6.1: *Barchart* sui *namespaces* e relative cardinalità

Dalla figura 6.1 si nota come la maggiorparte dei *namespaces* godano di più di una referenza al rispettivo autore. Inoltre è possibile accorgersi della comunque grossa quantità di *namespaces* **ambigui** con oltre 10 referenze. Proprio quest'ultimi rappresentano i candidati per l'applicazione di *Author Name Disambiguation*.

Infine con l'ultimo diagramma a barre è possibile vedere la cardinalità dei 10 *namespaces* più popolari in assoluto, che hanno cardinalità comprese fra le 80'000 e le 130'000 referenze.

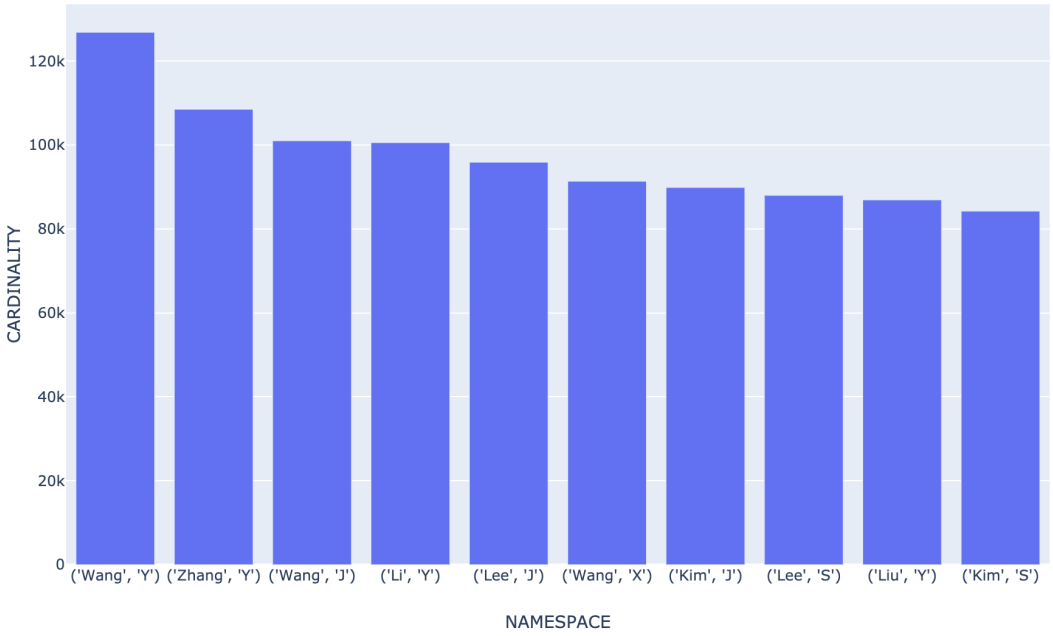

Figura 6.2: Diagramma a barre per i 10 *namespaces* più popolari

Questi sono i *namespaces* che più di tutti rendono il compito di disambiguazione nel database PubMed necessario ed arduo.

Bibliografia

[1] Dina Vishnyakova, Raul Rodriguez-Esteban, Khan Ozol, and Fabio Rinaldi. Author name disambiguation in medline based on journal descriptors and semantic types. *Association for Computational Linguistics*, 2016.

[2] V. I. Torvik and N. R. Smalheiser. Author name disambiguation in medline. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 2009.

[3] Mark-Christoph Müller. Semantic author name disambiguation with word embeddings. *Paper presentato alla International Conference on Theory and Practice of Digital Libraries*, 2017.

[4] H. Han, L. Giles, H. Zha, C. Li, and K. Tsioutsiouliklis. Two supervised learning approaches for name disambiguation in author citations. *Paper presentato al Digital Libraries*, 2004.

[5] P. Treeratpituk and C. L. Giles. Disambiguating authors in academic publications using random forests. *Paper presentato alla cerimonia del nono ACM/IEEE-CS joint conference on Digital libraries*, 2009.

[6] Y. Song, J. Huang, I. G. Councill, J. Li, and C. L. Giles. Efficient topic-based unsupervised name disambiguation. *Paper presentato alla cerimonia della settima ACM/IEEE-CS joint conference on Digital libraries*, 2007.

[7] Š. Dlugolinský, M. Ciglan, and M. Laclavík. Evaluation of named entity recognition tools on microposts. *Paper presentato nel 2013 alla 17-esima conferenza IEEE International Conference on Intelligent Engineering Systems (INES)*, 2013.

[8] Lenz Furrer, Anna Jancso, Nicola Colic, and Fabio Rinaldi. Oger++: hybrid multi-type entity recognition. *Journal of Cheminformatics*, 2019.

[9] S. M. Humphrey, C. J. Lu, W. J. Rogers, and A. C. Browne. Journal descriptor indexing tool for categorizing text according to discipline or semantic type. *Paper presentato al AMIA Annual Symposium Proceedings*, 2006.

[10] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *Paper pubblicato alla Neural Information Processing Systems Conference*, 2013.

[11] W. Cohen, P. Ravikumar, and S. Fienberg. A comparison of string metrics for matching names and records. *Paper presentato al Kdd workshop sul data cleaning ed object consolidation*, 2003.

[12] Leo Breiman. Random forests. *Springer*, 2001.

[13] Theodoros Evgeniou and Massimiliano Pontil. Support vector machines: Theory and applications. *Springer*, 1999.