UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTÀ DI SCIENZE E TECNOLOGIE

Corso di Laurea in Informatica per la Comunicazione

PROFILE MATCHING IN ONLINE SOCIAL NETWORKS

Relatore: Sabrina Gaito

Correlatore: Matteo Zignani

Tesi di:

Mattia Dimauro

Matricola: 808902

Anno Accademico 2013-2014

Ringraziamenti

Ringrazio Sabrina Gaito e Matteo Zignani, relatori di questa tesi, per avere reso possibile questo risultato e per la loro disponibilità nel seguirmi durante il lavoro. Voglio ringraziare inoltre Christian Quadri per il sostegno e la sua gentilissima disponibilità, e tutti i ragazzi di NPT Lab, per avermi ospitato e condiviso giornate di lavoro in laboratorio. Ringrazio Corrado Monti per tutto il contributo, i consigli e il gentile supporto mostrato in questi mesi. Grazie anche a tutte le persone di Tribe e Shaman tra cui Alessandro Egro, relatore aziendale, per la disponibilità ad ospitare questo progetto e per il vostro appoggio dimostrato. Ringrazio Giovanni Blu Mitolo e Martino Di Filippo, colleghi e amici, per l'importante supporto e per la loro generosità. Ringrazio infine i miei genitori, per il sostegno e l'affetto.

Indice

Capitolo 1

Introduzione

Lo spettro di servizi di reti sociali (*social network services*, SNS) presenti è ampio e variegato per dominio d'interesse, contenuti, tipologia di utenti e servizi offerti. Molte persone dispongono di un'eterogeneità di profili su diversi social network services per soddisfare i loro molteplici bisogni. Le informazioni presenti su ognuno di questi servizi possono differire, e quindi le informazioni riguardanti un individuo si ritrovano ad essere sparse su servizi diversi e spesso non facilmente riconducibili ad una singola fonte. Per poter integrare queste informazioni è necessario sapere identificare un individuo attraverso diversi social network services. La difficoltà riscontrata è dovuta in parte alla natura del web e all'azione della maggior parte dei SNS[1] che preservano l'anonimità dei loro utenti, permettendogli di scegliere liberamente uno username, uno pseudonimo, invece che la loro reale identità. A questo si unisce il fenomeno di utilizzare diversi username e sistemi di autenticazione sui diversi servizi che si utilizzano. Inoltre, nonostante si riscontri un crescente numero di servizi e utenti[10] che adottano tecnologie Single-Sign-On (SSO) come openID[2], dove gli utenti possono

[1] Social Network Services
[2] http://openid.net

autenticarsi utilizzando un singolo username (ad esempio, gli utenti possono autenticarsi su Google+ e YouTube utilizzando il loro account GMail), questi non coprono la totalità dei servizi e degli utenti.

Vi sono diverse applicazioni che beneficerebbero di un sistema di profile matching. Una applicazione è riscontrabile nell'area di marketing e business, dove una maggiore profondità di percezione e comprensione dei propri clienti, apporterebbe una migliore comunicazione e una più significativa espressione di messaggi con le persone. Strumenti di audience analysis come quello sviluppato recentemente da Facebook[1] permettono a venditori di apprendere maggiori informazioni riguardo il loro target, includendo informazioni aggregate riguardanti disposizione geografiche, demografiche, comportamento negli acquisti e altro. Questi servizi trarrebbero beneficio collegando gli individui tra comunità favorendo l'integrazione delle informazioni presenti sui diversi SNS. Una seconda applicazione, come alcuni studi hanno sottolineato[12], propone questo metodo come soluzione alternativa al problema di age verification. Se venissero fornite informazioni fittizie (ad esempio un'età incorretta), emergerebbero delle inconsistenze di informazione tra diversi SNS. Individuare queste inconsistenze può aiutare a fornire un primo approccio verso la risoluzione al problema di age verification. Una prima via per individuare queste inconsistenze è connettere le diverse identità di un utente attraverso diversi social network services. Per affrontare il problema di identificazione si potrebbero utilizzare una composizione di molteplici informazioni reperibili sui diversi SNS. Tendenzialmente si potrebbero usare tutti i dati a disposizione, che spaziano da quelli di profilo ad esempio username e avatar, dati personali anagrafici come nome, cognome, luogo e data di nascita o le connessioni con altri profili del SNS. Diversi di questi dati però potrebbero non essere disponibili sui vari servizi. Alcuni studi[7][12] hanno approcciato questo problema prendendo

in considerazione unicamente gli username. In termini di disponibilità delle informazioni, gli username sembrerebbero il denominatore comune disponibile su tutti i SNS. Solitamente uno username è formato da una serie di caratteri alfanumerici o da indirizzi email, grazie ai quali gli utenti riescono ad accedere al servizio. Questi sono tipicamente unici per ognuno dei SNS, mentre la maggior parte delle informazioni personali, anche la combinazione di nome e cognome non lo sono. Formalizzeremo il problema usando gli username come entità disponibile su tutti i servizi. Considerando gli username, ci sono due problemi che devono essere risolti per identificare un utente:

1. Dati due username, u_1 e u_2, riuscire a determinare se appartengono allo stesso individuo

2. Dato uno username u di un individuo I, trovare gli altri username di I

Il primo punto può trovare risposta in due passi: 1) troviamo l'insieme di tutti gli username C che probabilmente appartengono all'individuo I. Denotiamo l'insieme C come username candidate e, 2) per tutti gli username candidate $c \in C$, controlliamo se c e u appartengono allo stesso individuo. Quindi, se gli username candidate C sono noti, il problema 2 si riduce al problema 1. Possiamo rispondere al problema 1 trovando una funzione di identificazione $f(u,c)$

$$f(u,c) = \begin{cases} 1 & \text{se } c \text{ e } u \text{ appartengono allo stesso } I \text{ ;} \\ 0 & \text{altrimenti} \end{cases}$$

possiamo assumere noto che lo username u appartenga all'individuo I e considerare c il candidate username di cui vorremmo verificare l'appartenenza a I. In altre parole, u è l'informazione a priori (prior) fornita per I. La funzione di identificazione può essere generalizzata assumendo che l'informazione a priori sia un insieme di username $U = \{u_1,u_2,...,u_n\}$ e la nostra funzione di identificazione $f(.,.)$ diventerebbe

$$f(U,c) = \begin{cases} 1 & \text{se } c \text{ e l'insieme } U \text{ appartengono a } I \text{ ;} \\ 0 & \text{altrimenti} \end{cases}$$

dove U è l'informazione a priori che abbiamo di un individuo I, in questo caso un insieme di username, e c è lo username candidato di cui vorremmo testare l'appartenenza allo stesso I.

Questa tesi pone come obiettivi la progettazione e l'implementazione di una soluzione che consenta di stabilire la correlazione di identità tra profili di individui presenti su diversi social networks. Una prima fase del lavoro, come vedremo nel capitolo 2, consisterà nella implementazione di un algoritmo che conduca alla risoluzione del problema di profile matching individuando una serie di modelli comportamentali mostrati dagli utenti nello scegliere uno username che possono aiutare nell'identificazione di utenti tra differenti SNS. Implementeremo parte delle feature presentate da Zafarani et al[12] che catturano questi modelli e proporremo alcune feature addizionali, per poi discutere le differenze riscontrate nell'utilizzo di queste. Questo problema verrà affrontato utilizzando un dataset acquisito, come descritto nel capitolo 3, estraendo dati da fonti pubbliche raggiungibili su *internet* attraverso *web scraping* o *web data extraction*, una tecnica per l'estrazione di dati da pagine web. Verranno poi proposte e discusse nel capitolo 4, le soluzioni implementate per condurre l'analisi dei dati raccolti e l'implementazione di un modello predittivo, in particolare un modello di riconoscimento di pattern, capace di attuare predizioni accurate in base alle osservazioni fatte sui dati estratti descritti precedentemente.

Capitolo 2

Pattern comportamentali e costruzione di features

Gli individui mostrano spesso pattern comportamentali nella scelta dei loro usernames. Questi pattern, risultanti in ridondanza di informazione, possono essere utili per identificare individui su diversi social networks. I soggetti potrebbero evitare queste ridondanze selezionando username in modo che risultino completamente diversi dai loro altri usernames. In questa maniera gli username risulterebbero essere talmente differenti che, dato uno username, nessuna informazione riguardante gli altri usernames potrebbe essere estratta. Idealmente per raggiungere questo stato di indipendenza tra usernames, l'individuo dovrebbe scegliere uno username che presenti entropia massima. Ovvero uno username composto da una lunga sequenza di caratteri, lunga quanto il massimo consentito dal sistema, senza ridondanze: una sequenza di caratteri completamente casuale. Sfortunatamente, tutti questi requisiti non vengono incontro alle abilità umane. Gli esseri umani hanno difficoltà a memorizzare lunghe sequenze, con la possibilità della memoria a breve termine di ricordare 7 ± 2

elementi[6]. Queste limitazioni risultano condurre gli individui a selezionare generalmente usernames *non lunghi*, *non casuali* e che presentano *abbondante ridondanza*. Queste proprietà possono essere catturate adottando features specifiche.

Possiamo suddividere questi pattern comportamentali in tre categorie:

1. Pattern dovuti a limitazioni umane
2. Fattori esogeni
3. Fattori endogeni

Discuteremo dei comportamenti di ognuna di queste categorie elencate e delle features che possono essere estrapolate sfruttando questi pattern.
Riportiamo anche qui alcune notazioni già accennate nel capitolo 1 per chiarezza. Useremo spesso in questo capitolo, e nei capitoli a venire, notazioni quali "individui" , "username candidate", "prior username". Utilizzeremo la notazione *SNS* intendendo *Social Network Services*. Ci riferiremo a un "individuo" i, intendendo una persona umana, fisica, considerata nella sua singolarità. Utilizzeremo le parole "user" o "utente" per riferirci allo stesso concetto. Un individuo i, potrebbe essere registrato a diversi SNS, per il quale tendenzialmente è richiesto selezionare uno username u. Ci riferiremo all'insieme di tutti gli username conosciuti di i con la notazione U_i "priors username" o semplicemente "priors". Notare che U_i potrebbe essere un sottoinsieme di un insieme piú grande, con username addizionali, a noi però sconosciuti. Ci riferiremo invece a "username candidate" o semplicemente "candidate" o c_i intendendo uno username che si intende testare, ovvero capire se fa parte dell'insieme U_i. Riassumendo, abbiamo:

- I si riferisce all'insieme di individui i

- U_i riferisce all'insieme di username u per i o *priors username*
- c_i si riferisce al *candidate* per l'individuo i con priors U_i

2.1 Pattern dovuti a limitazioni umane

In generale, come umani, noi abbiamo:

1. Tempo e memoria limitati
2. Conoscenza limitata

Entrambe pregiudicano e influenzano i comportamenti nel selezionare i nostri username. Presentiamo qui le features individuate per catturare questi pattern comportamentali.

Username identici (1 feature) Uno studio recente[11] dimostra che il 59% degli individui preferisce usare lo stesso username reiteratamente, principalmente per facilità nel ricordarlo. Quindi se un candidate username c_i compare tra i priors username U_i, vi è una forte indicazione che che potrebbe essere associato allo stesso individuo i a cui sono associati i priors username. Considereremo dunque di utilizzare il numero di match esatti tra candidate username tra i prior usernames come feature.

Username Length Likelihood (6 feature) Allo stesso modo, gli utenti hanno tipicamente un insieme di potenziali usernames dal quale estraggono quando richiesto di crearne uno nuovo. Questi usernames hanno differenti lunghezze e dunque è possibile calcolarne una distribuzione. Consideriamo l_c la lunghezza del candidate username e l_u la lunghezza di uno username $u \in U_i$. Assumiamo che per ogni nuovo

username creato è probabile il verificarsi di

$$\min l_u \leq l_c \leq \max l_u$$

Ad esempio se un individuo è solito scegliere username di lunghezza 8 o 9 caratteri, è improbabile che questo consideri di crearne uno con lunghezze minori o maggiori. Considereremo quindi le lunghezza del candidate username e la distribuzione delle lunghezze dei prior usernames come features. La distribuzione verrà rappresentata da un numero fisso di features, descrivendola come

$$(\mathbb{E}[l_u], \sigma[l_u], \text{med}[l_u], \min l_u, \max l_u)$$

Unique username creation Likelihood (1 feature) Spesso gli utenti tendono a non creare nuovi username quando vogliono registrarsi a un SNS, ma preferiscono, se possibile, utilizzare uno username selezionato precedentemente. Siamo interessati a misurare lo sforzo che gli utenti sono disposti a mostrare nel creare nuovi username. Questo può essere approssimato dal numero di username unici (*uniq(U)*) su la totalità dei priors username U

$$uniqueness = \frac{|uniq(U)|}{|U|}$$

Uniqueness sarà una feature nel nostro insieme di feature. Si può pensare $1/uniqueness$ come la media di volte in cui un individuo utilizza uno username su diversi SNS prima di decidere di crearne uno nuovo.

2.2 Fattori esogeni

I fattori esogeni sono comportamenti osservati dovuti a influenze culturali o in generale dall'ambiente in cui l'utente si trova e vive. Possiamo dividere i fattori esogeni in due

categorie:

1. Pattern di battitura
2. Pattern linguistici

2.2.1 Pattern di battitura

Si può pensare alla tastiera come un vincolo imposto dall'ambiente. È stato mostrato[4] l'impatto che la configurazione dei tasti di una tastiera ha sulla scelta di passwords e username, "grazie" a una massiva falla di sicurezza[1], che ha permesso di accedere a 6.5 milioni di passwords di altrettanti utenti di LinkedIn. Ad esempio **qwer1234** e **aoeusnth** sono due password molto conosciute per essere selezionate da utenti che utilizzano tastiere QWERTY e DVORAK rispettivamente. La maggior parte degli utenti utilizzano tastiere con disposizione dei tasti conosciute come QWERTY e DVORAK (o qualche variante come QWERTZ o AZERTY)[8]
Le feature qui presentate, tentano di catturare questi schemi ricorrenti correlati alla tastiera. Si assume che le persone utilizzino entrambe le mani per digitare sulla tastiera e che siano a conoscenza della tecnica di dattilografia *touch typing*[9]. Elaboreremo queste feature per entrambe le configurazioni di tasti appena discusse.

Stessa mano (12 feature) Siamo interessati a calcolare la percentuale di tasti digitati utilizzando la stessa mano usata per il tasto precedente, nel digitare uno username. Questo valore è inversamente proporzionale alla inclinazione dell'utente a cambiare mano nel digitare.

[1] http://leakedin.org

Stesse dita (12 feature) Siamo interessati a calcolare la percentuale di tasti digitati utilizzando lo stesso dito utilizzato per digitare il tasto precedente.

Ogni dito (96 feature) Con queste feature vogliamo calcolare la percentuale di tasti digitati per ogni dito. Ad esempio 15% digitati con indice della mano sinistra, 2% con il mignolo della mano destra, ecc... . I pollici non vengono conteggiati, in quanto dovrebbero venire utilizzati solamente per il tasto barra spaziatrice.

Spostamenti verticali (48 feature) Per queste features vogliamo calcolare la percentuale di tasti digitati su ogni fila orizzontale di tasti presente sulla tastiera, dove la riga superiore contiene i tasti numerici e le tre file sotto si dividono il resto dei caratteri. Con queste feature si intendono catturare i movimenti orizzontali delle dita sulla tastiera nel digitare gli username.

Distanza percorsa Questa feature rappresenta la distanza approssimata (in metri), percorsa per digitare uno username assumendo che un tasto di una tastiera standard misura 1.8cm^2, inclusi i margini attorno al tasto.

Figura 1: Una heatmap che rappresenta un esempio di pattern riscontrabile su tastiera

Per ogni campione, elaboreremo ognuna di queste feature per lo username candidate e ognuno dei priors username. Per ottenere un numero di feature eguale per ogni sample, in quanto ciascuno potrebbe avere un numero differente di priors, comprimeremo i valori calcolati per ogni prior username utilizzando la tecnica mostrata in 2.1, che ne calcola una distribuzione.

2.2.2 Pattern linguistici

Oltre a fattori ambientali, alcuni fattori culturali come il linguaggio, influenzano la scelta di uno username. Gli utenti utilizzano spesso la stessa lingua o lo stesso insieme di lingue nello scegliere uno username. Dunque, ci si aspetta un risultato piuttosto coerente se si osservano le lingue di piú username appartenenti allo stesso individuo. Per individuare la lingua di uno username potremmo utilizzare un sistema sviluppato da terzi. Solitamente si tratta di sistemi che inferiscono la lingua di un testo utilizzando classificatori addestrati su n-grammi estratti da corpus di varie lingue. A seconda della fase di apprendimento di questi classificatori, questi potranno essere capaci di riconoscere un insieme di lingue diverse, ma rimane impegnativo poterle riconoscere tutte. Un'altra sfida sottoposta a questi classificatori si propone quando vengono analizzate parole che potrebbero non seguire pattern statistici di alcun linguaggio, ad esempio sostantivi o luoghi. Per queste ragioni la feature non verrà implementata, ma verrà proposto un metodo che affronta questo problema da una diversa angolatura in seguito.

2.3 Fattori endogeni

I fattori endogeni giocano un ruolo molto importante nella selezione di uno username. Alcuni di questi fattori sono da attribuire a:

1. Attributi personali (nome, età, sesso, ruolo, posizione lavorativa, ecc.)
2. Casualità
3. Usanze (abitudini, abbreviazioni, prefissi o suffissi etc.)

2.3.1 Attributi personali

Informazioni personali (156 feature) Come menzionato, una delle problematicità di un modello per l'identificazione della lingua a partire da uno username si riscontra nell'individuare la lingua di nomi specifici, come luoghi o altri aspetti specifici dell'individuo che sta selezionando lo username. Ad esempio non sarebbe semplice inferire la lingua di uno username **R2D2**[2] o **K2**[3]. Ad ogni modo, i pattern presenti in queste parole possono essere catturati analizzando la distribuzione delle lettere dell'alfabeto. Ad esempio un utente che sceglie lo username **piccettino**[4] il piú delle volte opterà per uno username con una distribuzione dell'alfabeto che presenta una probabilità doppia per le lettere 'c', 'i', 't' rispetto alle altre lettere. Dunque, calcoliamo la distribuzione dell'alfabeto per ambedue i candidate username e prior username da utilizzare come feature. Questa metodologia viene usata come alternativa al sistema di riconoscimento delle lingue. Inoltre, il beneficio nell'utilizzare la distribuzione delle lettere dell'alfabeto non è solamente il fatto di essere lingua-indipendente, ma anche

[2]R2-D2 è un personaggio robot nell'universo di StarWars
[3]Montagna del Pakistan
[4]Piccettino è un personaggio appartenente all'universo di Rat-Man

la capacità di carpire parole che hanno significato solo per l'utente.

2.3.2 Casualità

Casualità degli username (6 feature) Come indicato precedentemente se un utente selezionasse uno username completamente casuale, non genererebbe alcuna ridondanza di informazione. Possiamo quantificare la casualità degli username di un individuo e considerarla come feature che misura il livello di privacy dell'individuo, che potrebbe aiutare nell'identificarlo. Per misurare la casualità, considereremo l'entropia[2] della distribuzione delle lettere dell'alfabeto che compongono lo username candidato come feature. Questa verrà calcolata anche per ogni prior username, risultando in una distribuzione di valori di entropia usando la tecnica di distribuzione sopra mostrata in 2.1.

2.3.3 Usanze

Alcuni utenti mostrano delle usanze o abitudini che, come la famosa espressione ricorda, sono dure a morire e vengono puntualmente ripresentate. Abitudini comuni comprendono:

Modifica username Spesso gli utenti scelgono un nuovo username cambiando qualcosa del loro username precedente. Alcuni:

1. Aggiungono prefissi o suffissi, ad esempio **mattia** che diventa **mattia87**

2. Abbreviano i loro username, ad esempio **mattia.dimauro** che diventa **mattiadmr**

3. Alternano caratteri o inseriscono nuovi caratteri, come **mattia** che diventa **matt1a**

Le feature elencate qui sotto cercano di carpire queste modifiche.

Prefissi e suffissi (5 feature) Per individuare prefissi si può controllare se uno username è una sottostringa dell'altro. Dunque, si considera la lunghezza della massima sottostringa comune o LCS[5] come feature informativa di simiglianza tra tra candidate e prior.

Abbreviazioni (5 feature) Per individuare le abbreviazioni usiamo la lunghezza della massima sottosequenza[6], da non confondere con LCS, sensibile anche a lettere non consecutive che corrispondono in due stringhe. Questa viene calcolata per ogni prior username e ne viene elaborata la distribuzione.

Caratteri invertiti o nuovi inserimenti (10 feature) Per individuare i caratteri invertiti o aggiunti faremo uso di alcune metriche di similarità su stringhe. Queste sono tecniche per quantificare la dissimilarità tra due stringhe. In particolare utilizzeremo la distanza di Levensthein, una metrica per misurare la differenza tra due sequenze, e l'indice di Jaccard, un indice statistico utilizzato per comparare la similarità e la diversità di insiemi campionari. L'indice di Jaccard per misurare la similarità

[5]Longest Common Substring

[6]Longest Common Subsequence Utilizzato in bioinformatica per analizzare sequenze di DNA, e alla base del tool *diff* per comparare testi

degli username non viene utilizzato come feature nel lavoro di Zafarani, a noi sembrava opportuno utilizzarla in quanto identificativa sul problema di classificazione affrontato, come vedremo meglio nel capitolo seguente.

Capitolo 3

Collezione del dataset

Esistono più modalità attraverso le quali è possibile collezionare dati riguardanti stessi individui presenti su diversi social networks. Un modo semplice consisterebbe nel organizzare un questionario dove si richieda agli utenti di elencare i propri profili. Questo metodo permette di raccogliere una quantità di dati spesso limitata. Esistono compagnie, alcune risultano essere gli stessi social networks, che richiedono queste informazioni ai propri utenti ma queste non sono disponibili pubblicamente. Fortunatamente esistono servizi di *social network aggregation* che permettono di collezionare contenuti da diversi *social network services* in una presentazione unificata. Il compito svolto da un *social network aggregator*, che raggruppa insieme informazioni in un singolo luogo, supporta i propri utenti a riunire molteplici profili di social network in un singolo profilo e aiuta a tenere traccia delle attività che avvengono sui diversi profili semplificando la *social networking experience* dell'utente. La maggior parte di questi servizi non permette l'accesso pubblico alle informazioni aggregate, sono per lo più uno strumento per l'utilizzatore per seguire i propri profili, permettendo di avere tutte le notifiche relative in unico luogo, o di pubblicare lo stesso contenuto in più

profili in una volta sola. Questa tipologia di aggregatori non sono frutto di interesse per la collezione di informazioni che cerchiamo. Prenderemo in considerazione invece quegli aggregatori che permettono di condividere queste informazioni aggregate con altre persone. Attraverso questi è possibile visualizzare tutte le attività di un utente sui diversi social network che questo a deciso di far seguire all'aggregatore. Ovviamente è possibile, ed è l'informazione che andremo a estrarre, risalire su quale social network è stata eseguita tale azione e avere così un elenco di profili di social network services appartenenti alla stessa persona.

3.1 Scraping di Alternion

Alternion - All your social web and email in one place - è un aggregatore i cui servizi e features lo fanno rientrare tra la seconda classe di *social network aggregator* descritta. Attraverso *Alternion* è infatti possibile per un utente condividere con altri contatti le informazioni riguardanti i propri profili di diversi online social networks. Il numero di utenti iscritti al servizio non è noto. Alternion dichiara di permettere ai propri utenti di aggregare un grosso numero di online social networks, dai più popolari come *Facebook*, *Twitter*, *Google+*, *LinkedIn*, *Flickr* fino a più di 220 online social networks. Di seguito spiegheremo l'approccio utilizzato al fine di ottenere i profili degli utenti iscritti al servizio e di recuperare da questi le informazioni riguardati i profili dei loro social networks. Alternion non dispone di un servizio per recuperare i dati attraverso un'interfaccia *web (API)*, con richieste e risposte documentate. Dovremo dunque procedere analizzando le pagine web restituite svolgendo un'attività conosciuta come *web scraping*. Il *web scraping* o *web data extraction* è una tecnica di estrazioni di dati da pagine web. L'estrazione di dati viene automatizzata attraverso un software

che simula un utente umano nell'esplorazione di documenti presenti sulla rete internet. Il software deve quindi implementare il protocollo HTTP, fondamento per la comunicazione di dati per il World Wide Web per recuperare il documento attraverso la rete internet. Questi documenti, tipicamente descritti attraverso un linguaggio di markup, sono pensati per rappresentare un'interfaccia grafica per l'utente. Ottenuto il documento il *web scraper* si occupa di analizzarne i dati strutturati ricercandone quelli di interesse. La pagina non viene interpretata visivamente ma esaminandone il contenuto descritto con il linguaggio di markup, tipicamente HTML. Questa ricerca può essere effettuata con diversi approcci: dal più semplice, seppur potente, *text grepping* combinato con *regular expression matching* o può prevedere un'analisi più strutturata della pagina attraverso una tecnica di *DOM parsing*. Per i nostri scopi utilizzeremo entrambe queste tecniche. Esistono altre tecniche e metodologie per eseguire *web scraping*, ma i dettagli esulano dallo scopo di questa tesi.

3.1.1 Ottenere i profili

Siamo interessanti a ottenere un considerevole numero di profili di utenti che utilizzano l'aggregatore in analisi. Alternion non prevede una funzionalità per mostrare l'elenco completo dei profili iscritti al proprio servizio. Permette invece di reperirne un sotto insieme di questo attraverso una funzionalità di ricerca, divisibile in due classi: la ricerca parametrizzabile secondo alcuni criteri o la presentazione casuale di profili. La prima permette di interrogare il sistema per estrarne i profili corrispondenti ad alcuni parametri di ricerca come Nome, Sesso, Età, Paese di provenienza, Educazione, Interessi, etc. La seconda funzionalità consente di ottenere ad ogni richiesta un profilo selezionato casualmente.[1] Ho deciso di accantonare questa seconda via per due

[1] Più probabilmente, pseudo-casualmente

ragioni. La selezione casuale potrebbe potenzialmente portare a richiedere più volte lo stesso profilo, dipendentemente dalla bontà (che non è stata testata) della casualità con cui il profilo viene estratto. Vogliamo inoltre limitare l'estrazione di profili non appartenenti a persone fisiche: alcuni profili presenti fanno infatti riferimento a prodotti o società e aziende. Concludo di optare per la ricerca parametrizzata, in particolare, la ricerca per nome. Come lista di parametri per l'interrogazione del sistema useremo una lista di nomi propri [2], formata da 4275 nomi femminili e 1219 maschili. Utilizzando strumenti messi a disposizione da Google[3] per analizzarne i pacchetti HTTP scambiati tra il server di Alternion e il client Google Chrome, viene identificata la richiesta HTTP da eseguire per interrogare il server per farsi restituire la pagina web contenente la lista di utenti corrispondenti al parametro di ricerca. Eseguiremo quindi una richiesta per ogni nome presente nelle nostre liste di nomi. Da questa, tramite tecniche di scraping descritte precedentemente, estrapoliamo per ogni utente l'*URL* identificativo della risorsa. Una volta collezionati tutti gli URL, dove ogni indirizzo corrisponde a un utente di Alternion, potremo recuperare la pagina profilo di questi utenti e cercare al suo interno le informazioni riguardanti i loro social networks. Per la persistenza dei dati useremo MongoDB, un database NoSQL document-oriented, che utilizza JSON come data model.

3.1.2 Profili recuperati

Sono stati recuperati 15341 profili di Alternion, di cui 11274 presentano almeno due profili di social networks. Il numero di profili non è ingente, in quanto la funzionalità ricerca permette di recuperare un massimo di 30 profili per richiesta, ad esempio solo trenta profili delle persone che si chiamano 'Anna'. Ad ogni modo si è notato che

[2]Lista reperita da census.gov
[3]Google Chrome DevTools

è possibile scalare, se non verticalmente per numero di users, orizzontalmente per numero di profili per users. Ogni utente ha in media ∼4,6 OSNS collegati, distribuiti tra un totale di 168 online social network services diversi presenti. Possiamo dunque espandere il numero di coppie di usernames secondo una combinazione semplice di n elementi di classe k, dove $n = 2$ (coppie) e k il numero di classi di OSN distinti. Ad esempio, un profilo P ha aggregato 3 OSNs {Facebook, Twitter, Instagram } e presenta quindi uno username u per ogni profilo $U = \{\mathrm{u_{AL}}, \mathrm{u_{FB}}, \mathrm{u_{TW}}, \mathrm{u_{Instagram}}\}$.

I sottoinsiemi di cardinalità 2 dell'insieme U sono:

- $\{\mathrm{u_{AL}}, \mathrm{u_{FB}}\}$
- $\{\mathrm{u_{AL}}, \mathrm{u_{TW}}\}$
- $\{\mathrm{u_{AL}}, \mathrm{u_{INM}}\}$
- $\{\mathrm{u_{FB}}, \mathrm{u_{TW}}\}$
- $\{\mathrm{u_{FB}}, \mathrm{u_{INM}}\}$
- $\{\mathrm{u_{TW}}, \mathrm{u_{INM}}\}$

Potenzialmente, il numero di classi di coppie di social network possibile è uguale al coefficiente binomiale

$$\binom{n}{k} = \frac{n(n-1)\dots(n-k+1)}{k(k-1)\dots 1} = \frac{n!}{k!(n-k)!}$$

dove $k = 2$ (coppie) e $n = 168$, quindi 14028 coppie di social network distinte. Solamente 5855 di queste però presentano almeno una coppia di username al suo interno. Concludendo, applicando la combinazione a 2 elementi per ogni profilo nel nostro

dataset, otteniamo ∼170000 coppie di usernames. Con una media di ∼29 coppie di username per classe. In seguito presenteremo le distribuzioni delle distanze, secondo alcune metriche che mostreremo, tra le coppie di username appena generate per analizzare la similarità degli username di uno stesso individuo.

OSNS presenti {100zakladokru, 43 Things, 500px, ActiveRain, All Consuming, Alternion, Amazon, Ameba, Aminus3, Answerbag, Aol Answers, AudioBoo, Bambuser, Bebo, Blipfm, Blipfoto, Bliptv, Blog Talk Radio, Blogger, Blogmarks, Blogru, Blogs@MailRu, Bordom, BuzzFeed, Buzznet, CafeMom, CiteULike, Connotea, Current, DailyStrength, Dailymotion, Delicious, DeviantART, Diigo, Disqus, Docstoc, Douban, Dreamwidth, Dribbble, EmpoweHER, Etsy, Eventbrite, FFFFound!, Facebook, Fancy, Flickr, FoodFeed, Formspring, Fotolog, Foursquare, FunnyOrDie, GamerDNA, Gamespot, Gather, GitHub, Gizmodo, Goodreads, Google Reader, Google+, Habrahabr, Hatena Bookmark, Hatena Diary, Hatena Haiku, HubPages, Hyves, Identica, Imgly, Instagram, Instructables, IntenseDebate, Ipernity, Issuu, Jalbum, JamBase, Judy"s Book, Lafango, Lastfm, LibraryThing, LinkedIn, Listal, Lookbooknu, Magma, MeasuredUp, Memoriru, Meneame, MetaFilter, Metacafe, Mister Wong, Moblog, MobyPicture, Multiply, Netlog, News2ru, Newsvine, NowPublic, Pandora, Panoramio, Photobucket, Photocase, Photosightru, Picasa, Pikchur, Pinboard, Pinterest, Plancast, Plixi, Plurk, Polyvore, Posterous, Qik, Qype, RPodru, Raptr, RedBubble, RedGage, Reddit, Rooftop Comedy, SAPO Fotos, SAPO Videos, Server Fault, Six Groups,Skyrock, SlideShare, SmugMug, Soupio, SparkPeople, Squidoo, Stack Overflow, StumbleUpon, Super User, Tabulas, Technorati, ThisNext, Threadless, TravelPod, Trilulilu, Tripit, Trulia, Tumblr, Tvigleru, Twitgoo, Twitpic, Twitrpix, Twitter, UserVoice, Viddler, VideoQip, Vimeo, Wattv, We Heart It, WordPress,

Worth1000, Xanga, Yahoo! Answers, YouTube, Zazzle, Zenfolio, Zillow, Zorpia, aNobii, authorSTREAM, Facebook, gdgt, I use this, iPadio, iReport, Visualizeus, wePapers}

Lo spettro di social networks presenti è ampio e variegato per dominio d'interesse, contenuti, tipologia di utenti e servizi offerti. Non è semplice crearne una tassonomia, che in ogni caso, non delineerebbe dei gruppi chiaramente separati e privi di sovrapposizioni. Notiamo la presenza dei SN piú popolari come Facebook, Twitter, Google+, passando per sistemi conosciuti per le loro funzionalità di *Location Based Services* (LBS) come Foursquare, o sistemi distinti per contenuti a tema musicale, fotografico o video come Lastfm, Pandora, Flickr, Instagram, 500px, Youtube, Vimeo, e ancora, sistemi dedicati al blogging come Tumblr, o sistemi di social bookmarking come StumbleUpon e altri.

3.1.3 Distribuzione della similarità degli username

Siamo interessati a confrontare coppie di usernames, per tentare di ricollegarle alla stessa entità. Il caso piú semplice che potrebbe presentarsi[4] è la corrispondenza esatta tra i due usernames. Indagheremo quindi questa proprietà sul nostro dataset acquisito. Oltre a verificarne la corrispondenza esatta, ovvero che le due stringhe che formano gli username presentino gli stessi caratteri nello stesso ordine, faremo uso di alcune metriche di similarità su stringhe. Queste sono tecniche per quantificare la dissimilarità tra due stringhe. In particolare utilizzeremo la *distanza di Levensthein*, una metrica per misurare la differenza tra due sequenze, e *l'indice di Jaccard*, un indice statistico utilizzato per comparare la similarità e la diversità di insiemi campionari.

[4]se non si considerano i casi di omonimia

Distanze Riportiamo le statistiche sulle distanze calcolate sulle classi aventi almeno 1000 coppie di username (29 classi, con un totale di 56687 coppie di usernames) e sui dati globali.

	Mean Levensthein distance	Mean Jaccard Index
Popular classes	6.2295	0.3277
All dataset	5.8159	0.3685

Match Esatti Qui riportiamo le statistiche su i match esatti, dove le distanze di Levensthein e l'indice di Jaccard assumomo il valore 0.

	Levensthein distance = 0	Jaccard Index = 0
Popular classes	31.79%	34.03%
All dataset	27.80%	29.16%

Riportiamo alcuni grafi per illustrare alcune proprietå dei dati raccolti.[5] In Fig1 e Fig2 vengono riportati gli istogrammi che rappresentano la distribuzione delle distanze di Levenshtein e dell'indice di Jaccard rispettivamente tra le coppie di username del dataset. L'istogramma in Fig1 riporta sull'asse x le distanze e sull'asse delle y il numero di coppie che ricade in quella classe. Il numero di classi e il suo spettro riflette le distanze riscontrate per ogni coppia: da 0, distanza minima oltre la quale non è possibile scendere a 45 massima distanza presente tra due username in tutto il dataset. Per l'istogramma in Fig2, il numero di classi o *bin* è fisso, con un range 0-1, ovvero lo stesso spettro di valori assumibili dall'indice di Jaccard. Notiamo in

[5]Tutti i grafi qui riprodotti sono stati generati con l'ausilio di matplotlib[6], una popolare libreria open source per rappresentare set di dati con grafi bidimensionali come plots, istogrammi e bar charts, descrivibili attraverso poche linee di codice, in specifico, Python.

entrambi i grafi di Fig1 e Fig2 una distribuzione multimodale, è abbastanza semplice infatti individuare due picchi nei due istogrammi. Il primo picco, sullo 0, è facilmente riconducibile alle coppie di username identici tra di loro.

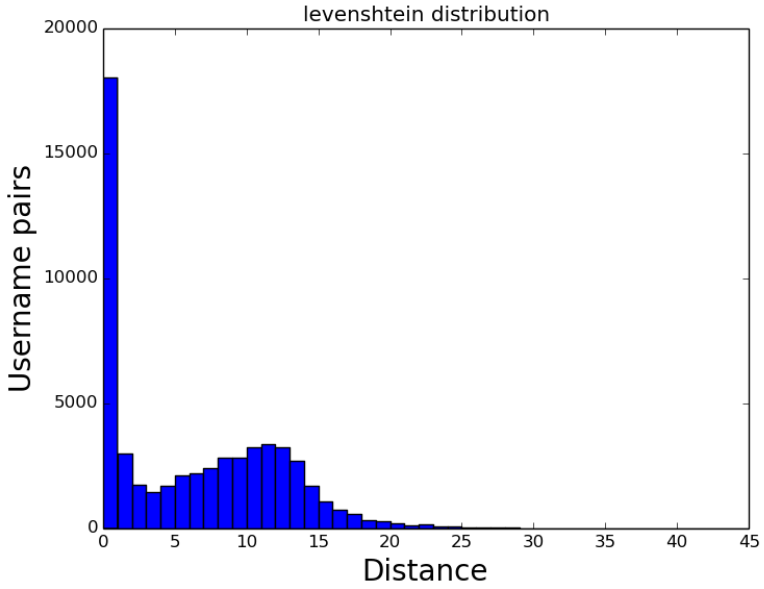


Figura 2: Distribuzione della distanza di Levensthein tra coppie di usernames

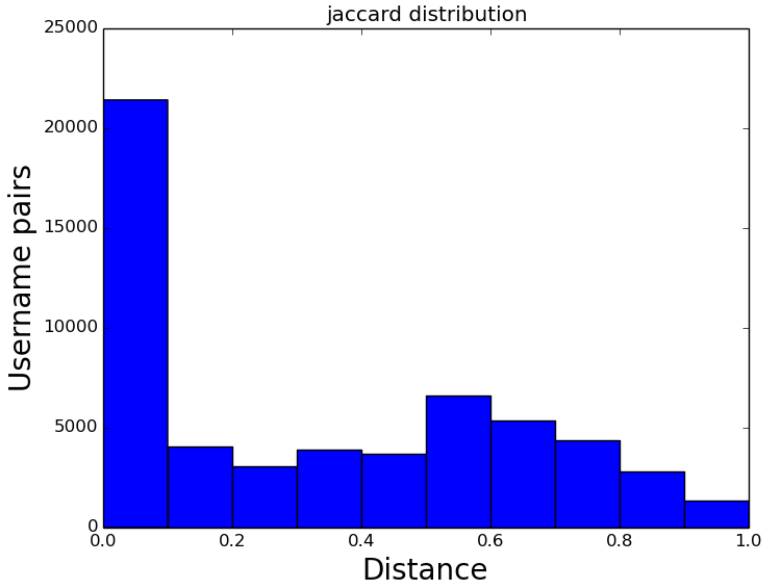


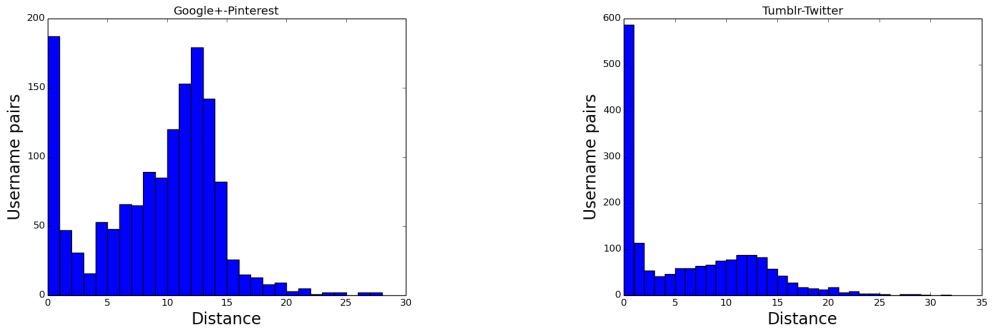
Figura 3: Distribuzione indice di Jaccard tra coppie di usernames

In Fig3 mostriamo come i dati riportino un pattern a *doppio picco* sia su dati globali che su alcune classi prese singolarmente.

In Fig4 e Fig5 riportiamo la distribuzione delle distanze tra coppie di username casuali, non appartenenti alla stessa persona. Il mescolamento delle coppie di username è ottenuto tramite una tecnica usualmente riferita come *convolution* o *zip function*, una funzione che mappa una tupla di sequenze in una sequenza di tuple. Se ad esempio si ha una lista di 3 coppie/tuple di username u per utenti i:

$$[(i_1u_1, i_1u_2), (i_2u_1, i_2u_2), (i_3u_1, i_3u_2)]$$

otteniamo, tramite l'inverso della funzione descritta prima (*unzip*), una lista con due tuple:



(a) Google+ Pinterest (b) Tumblr Twitter

Figura 4: Distribuzione multimodale, con pattern a doppio picco, anche su classe singola

$$[(i_1u_1, i_2u_1, i_3u_1), (i_1u_2, i_2u_2, i_3u_2)]$$

A questo punto, mescoliamo in maniera casuale l'ordine degli elementi nelle due tuple, e attraverso la funzione zip ricombiniamo le due tuple, ottenendo cosi coppie di username non appartenenti allo stesso utente individuo i. Ad esempio potremmo ritrovare:

$$[(i_1u_1, i_2u_2), (i_2u_1, i_3u_2), (i_3u_1, i_1u_2)]$$

Applichiamo questa tecnica alle coppie di username del nostro dataset, e calcoliamo la distribuzione delle distanze in maniera analoga a quella presentata precedentemente. Quello che riscontriamo è, una piú prevedibile, distribuzione normale (Gaussiana), come si può notare in figure 4 e 5.

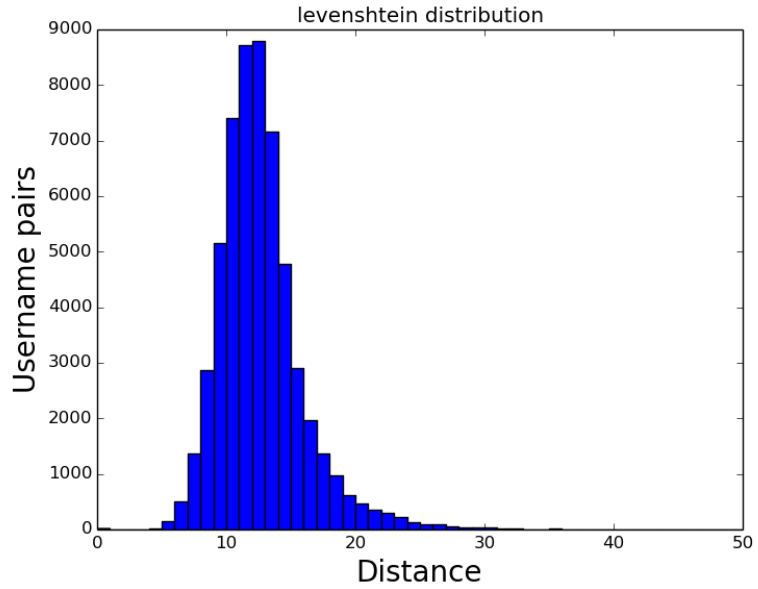


Figura 5: Distribuzione unimodale distanza di Levensthein tra coppie random

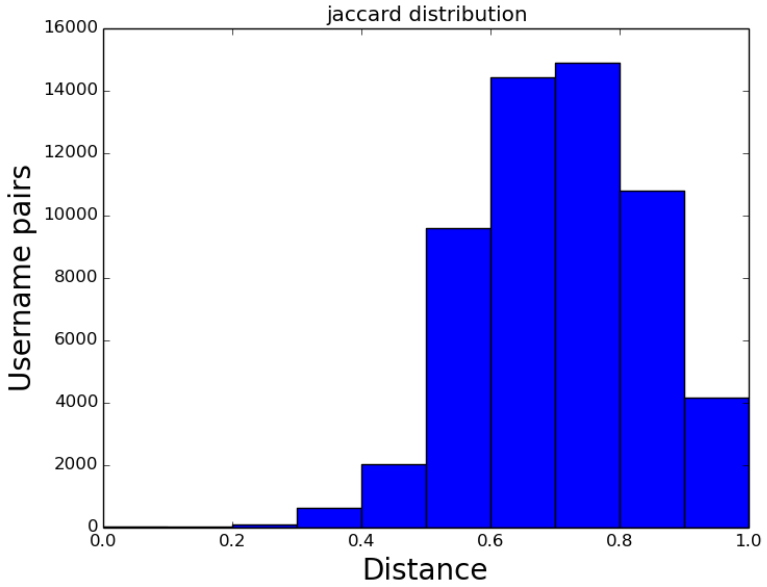


Figura 6: Distribuzione indice di Jaccard tra coppie random

Capitolo 4

Apprendimento e Risultati

La metodologia proposta nei precedenti capitoli verrà sistematicamente valutata in questo capitolo. Mostreremo i risultati ottenuti utilizzando le features individuate nel secondo capitolo, con i dati raccolti come esposto nel capitolo precedente. In Fig 2 riporto un diagramma di flusso redatto da Sebastian Raschka, il quale dà un'ottima visione globale della la pipeline di processi affrontati nell'apprendimento supervisionato e quindi quelli di questo capitolo. Come metriche per misurare la bontà del modello di classificazione useremo, l'Accuracy (ACC), la F-measure (F1), e la Area Under Curve (AUC) della funzione Receiver Operating Characteristic (ROC), brevemente illustrate qui sotto. Uno strumento tipico per valutare la performance è la così conosciuta "matrice di confusione", una matrice quadrata che consiste in colonne e righe che descrivono il numero di istanze come il rapporto tra "classe attuale" e "classe predetta". Una semplice matrice di confusione per il classico problema di classificazione di "spam vs. ham", un classificatore che individua tra le email ricevute quelle che sono da considerare posta indesiderata, potrebbe essere quella riporta in Fig1.

L'accuratezza è definita come la frazione della classificazioni corrette su il numero totale di samples. È calcolata come:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

dove TP= True Positive, TN= True Negative, ovvero le istanze classificate correttamente, FP = False Positive, il numero di istanze classificate erroneamente positive, e FN = False Negative, il numero di istanze classificate erroneamente negative. Altri indicatori per le performance di un classificatore sono la Sensibilità o Recall e la Precision. Che si calcolano in questo modo:

$$Recall = \frac{TP}{TP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

La F-measure prende in considerazione sia la Precision che la Recall. Questa può essere interpretata come la media armonica pesata della Precision e la Recall. Spesso viene utilizzata la F1-measure o *balanced F-score* (F1 score), questa è la media armonica pesata con pesi unitari e si ottiene calcolando:

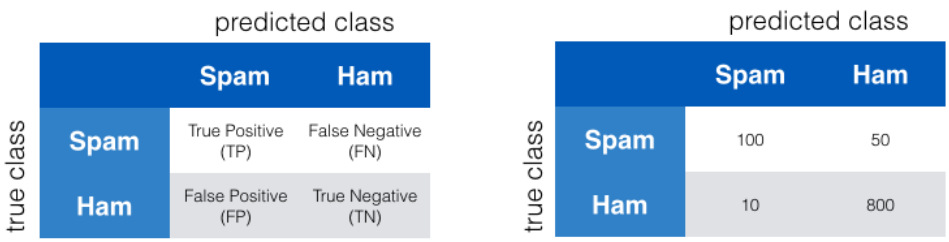


Figura 1: Matrice di confusione Spam vs Ham

$$F1 = 2 \cdot \frac{TP \cdot FP}{TP + FP}$$

4.1 Preprocessamento dei dati

Per ogni profilo P con un insieme di username U, vogliamo creare una tupla:

$$(candidate, priors)$$

dove *candidate* vale u e *priors* vale $u \setminus U$, per ogni usernames u in U.
Prima di estrarre le features dai samples del nostro dataset, dobbiamo preprocessare i dati acquisiti in modo da non avere sample con dati mancanti. Applicando le seguenti politiche:

- l(*candidate*) <1 ⇒ sample eliminato
- l(*prior*) <1 ⇒ prior rimosso dalla lista di priors
- l(*priors*) <1 ⇒ sample eliminato

dove l è una funzione che ritorna la lunghezza di una stringa di caratteri o la cardinalità di un insieme di stringhe, otteniamo un dataset pulito che permetterà di evitare errori nella fase di estrazione delle features.
I dati ottenuti rappresenteranno l'insieme dei nostri samples che verranno etichettati di classe positiva, ovvero samples di cui si riconosce l'attribuzione di ogni usernames verso un individuo I. Un classificatore binario ha peró bisogno anche di samples etichettati con una seconda classe. Nel nostro caso siamo interessati a separare usernames associati a uno stesso individuo da quelli che non lo sono. Dovremmo quindi

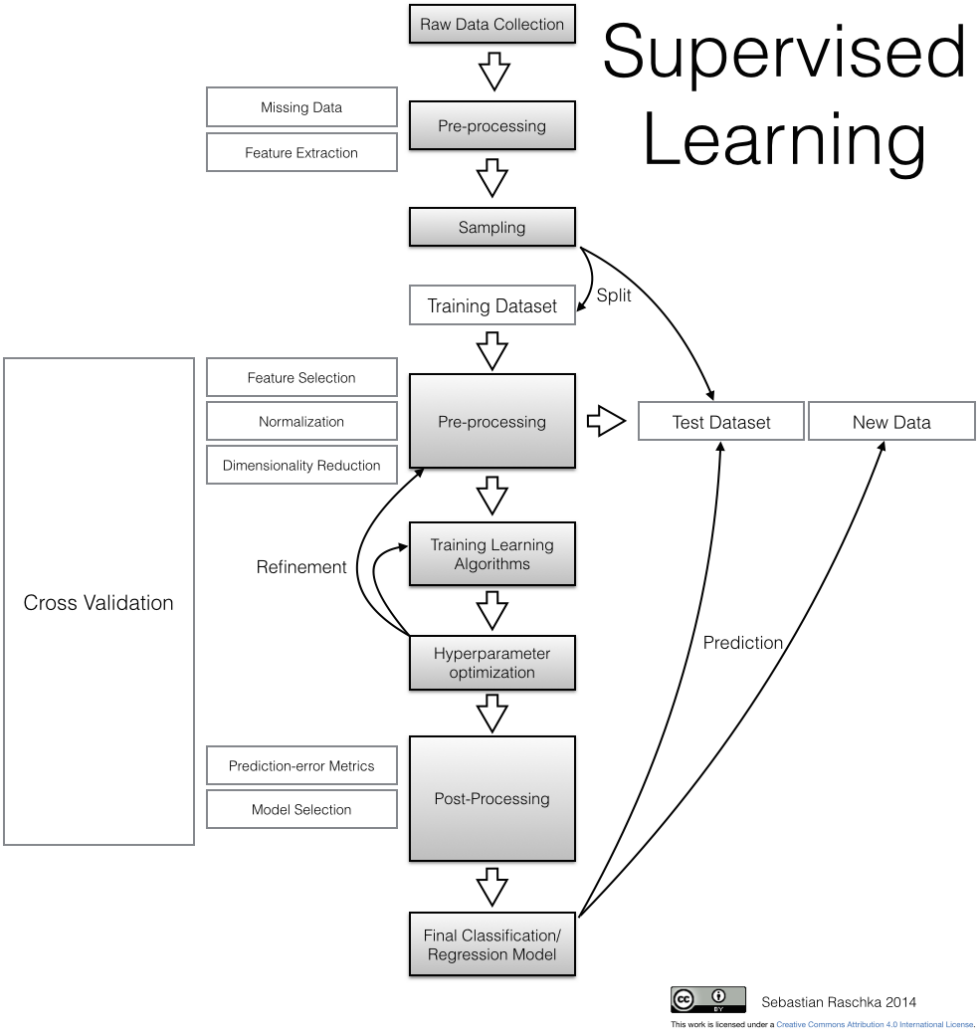


Figura 2: Supervised learning flowchart

elaborare a partire dal nostro dataset un insieme di samples che verrano etichettati come esempi negativi, ovvero samples che non godono della proprietà elencata prima per i samples positivi. Otteniamo questi samples tramite una tecnica usualmente riferita come *convolution* o *zip function* con un procedimento del tutto simile a quello affrontato 3.1.3 per il calcolo della distribuzione della distanza di Levensthein tra coppie random. Prima di poter utilizzare i dati per l'algoritmo di apprendimento, bisogna ulteriormente processarli. Allo stato attuale, i nostri dati sono ancora in forma testuale, ovvero sequenze di caratteri. Per risolvere il problema di classificazione, bisogna prima trasformare i dati in vettori di numeri reali, dove ogni numero rappresenta una feature.

4.1.1 Estrazione di feature

La trasformazione dei samples in vettori di features avviene applicando le funzioni viste nel capitolo 2. Queste, oltre a catturare gli aspetti descritti nel capitolo 2, restituiscono una rappresentazione dei dati utile al risolvimento del problema di classificazione. Le funzioni ritornano infatti un vettore di numeri di dimensione n, rappresentabile come un punto in uno spazio n-dimensionale. Rappresentando ogni sample come un punto in uno spazio, questo rende possibile la separazione dei samples in due classi, separando lo spazio in due semi-spazi, mediante un iper-piano di dimensione n-1.

4.1.2 Sampling

Divideremo ora il nostro dataset in maniera casuale in due parti, una di training e una di test. Il dataset di addestramento verrà utilizzato per addestrare il nostro modello, mentre lo scopo del dataset di test è quello di valutare le performance del

modello finale. È importante non usare il dataset di test nella fase di addestramento per evitare *overfitting* nel calcolo delle metriche di errore di predizione. L'overfitting porta a classificatori che danno ottimi risultati sui dati di addestramento ma non generalizzano bene il problema, risultando inefficienti e con un tasso di predizione-errore elevato quando vengono mostrati pattern/dati non ancora visti.

4.1.3 Normalizzazione dei dati

La normalizzazione o altre tecniche di *feature scaling* sono spesso mandatorie per ottenere risultati apprezzabili, anche se alcuni modelli di classificazione sono più sensibili di altri alla normalizzazione delle feature. Con il termine "normalizzazione" si intende spesso il portare gli attributi ad assumere un certo spettro di valori, ad esempio tra 0 e 1. Un altro approccio è il processo conosciuto come "standardizzazione" o *z-score* dove si prevede di sottrarre ad ogni campione la sua media e dividere il tutto per la deviazione standard in modo da ottenere le proprietà di una distribuzione normale, ovvero varianza pari a 1 e media pari a 0.

4.2 Addestramento del classificatore

Useremo ora i dati ottenuti con il nostro algoritmo di apprendimento per valutarne l'efficienza. Complessivamente, il dataset conta di ∼103000 sample, divisi equamente tra etichettati positivamente e negativamente. Testeremo il classificatore prima su la complessività dei dati e in seguito tenteremo alcuni diversi approcci per valutarne l'andamento al variare di alcuni fattori, in particolare:

- Il numero minimo di priors username
- Le classi di features utilizzate (*features selection*)

4.2.1 Variare il numero di priors username

Tra i samples del nostro dataset il numero di priors sono distribuiti come riportato in Fig2. Da un minimo di 1 prior username, dove troviamo ∼18000 sample fino a un massimo di 26 prior username, con molti meno samples a disposizione. Osserveremo l'andamento della accuratezza del classificatore al variare del numero di priors conosciuti.

4.2.2 Variare le classi di social networks

Come visto in 3.1.2 esistono 5855 coppie di social networks che contengono almeno una coppia di username, e il numero di situazioni possibili aumenta ora che stiamo considerando di variare il social network di partenza (dello username candidate) e quello degli username priors, che può essere più di uno. Testeremo empiricamente variando e filtrando sia username candidate che username priors per i social networks ritenuti più popolari.

4.2.3 Variare il numero di features - Features selection

Utilizzando tutte le classi e funzioni di features viste nel secondo capitolo si ottiene un vettore di 358 features. Restringeremo le classi di features utilizzate per l'algoritmo di apprendimento analizzando le performance del classificatore al variare di queste. Proveremo in un primo momento a limitare le classi di features applicate empiricamente, per analizzarne i risultati. Eseguiremo in seguito un'approccio di *feature selection*, in particolare utilizzeremo *Principal Component Analysis* o PCA. Lo scopro principale dei due approcci è quello di rimuovere il rumore e aumentare l'efficienza computazionale mantenendo solamente le informazioni "utili" e discriminatorie, per evitare

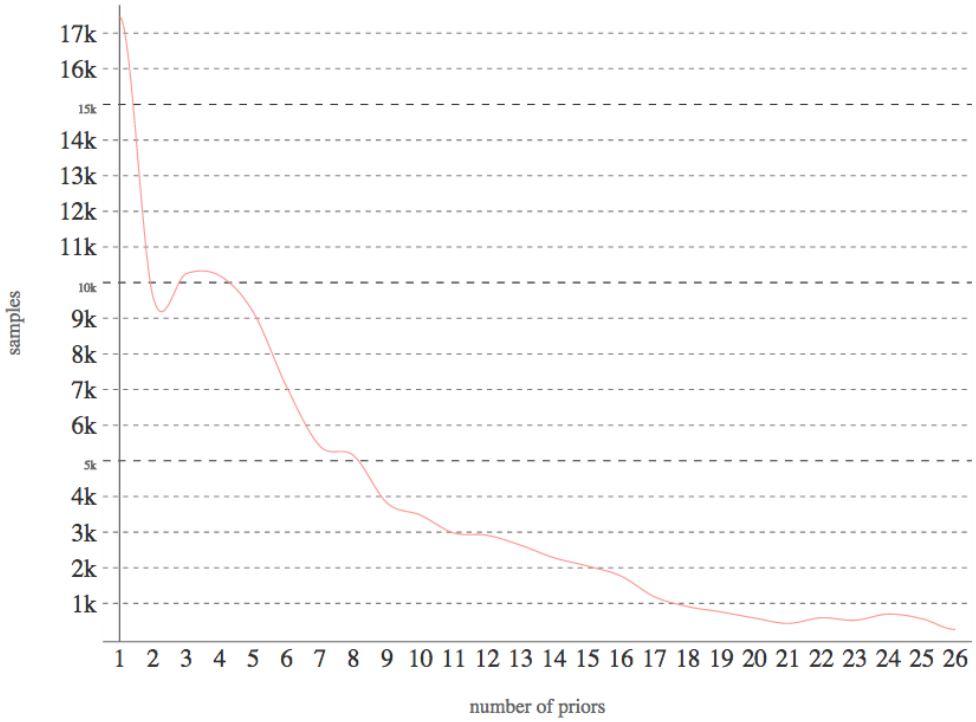


Figura 3: Numero di campioni per numero di prior username

overfitting o il problema noto come *curse of dimensionality*. Riferendoci alle features descritte nel secondo capitolo, assegnamo un'etichetta ad alcuni sottoinsiemi di queste e mostriamo i risultati ottenuti utilizzando questi insiemi limitati di features:

- ALL - Tutte le features
- K - DVORAK layout keyboard Exogenous features
- Q - QWERTY layout keyboard Exogenous features
- X - All Exogenous features (K+Q)
- E - Endogenous features
- H - Human Limitations features
- D - Distances features

4.3 Modelli di classificazione e risultati

Per l'implementazione dei modelli di classificazione utilizzati in questo capitolo si fa riferimento alla libreria SciKit Learn[1], un insieme di moduli Python per machine learning e data mining.

In fase di sviluppo, quando ancora non tutte le features erano state realizzate e avevamo un insieme molto limitato del dataset attuale, utilizzavamo Support Vector Machine (SVM) come modello di apprendimento. Le versioni che davano i risultati migliori erano quelle che utilizzavano un kernel RBF. Ad ogni modo, affrontando il problema utilizzando SVM senza kernel, portava a performance simili, indicandoci che probabilmente il problema era risolvibile linearmente. Con il crescere del numero

[1]"scikit-learn - Machine Learning in Python", http://scikit-learn.org

delle features e dei samples a disposizione, crescevano anche i tempi di esecuzione dell'algoritmo e la memoria richiesta dal calcolo, diventando troppo onerosi. È nata così la necessità di passare a un modello di classificatori diversi, conosciuti come "online", che hanno permesso di affrontare il problema con risorse di tempo e memoria molto più limitate. Non escludiamo che, a fronte di potenza di calcolo e memoria adeguata, SVM avrebbe potuto performare meglio dei modelli di classificazione utilizzati di seguito.

4.3.1 Classificatori binari online

Quello che definisce l'apprendimento online è il modo in cui gli esempi del training set vengono presentati all'algoritmo. Se infatti gli algoritmi di apprendimento fanno in generale l'assunzione di poter vedere il dataset interamente, quelli online hanno la limitazione di poter vedere un esempio alla volta, generalmente applicando questo schema ripetutamente:

1. Viene osservata un'istanza
2. L'algoritmo predice un'etichetta per l'istanza che può essere +1 o -1
3. Viene rivelata la vera etichetta
4. L'algoritmo subisce una perdita[2] che riflette il grado di errore della predizione appena eseguita, e aggiorna il suo stato interno.

Questo modo di agire presenta alcuni vantaggi:

[2]Secondo una funzione di perdita, conosciuta come *loss hinge function*

- Gli algoritmi online possono essere addestrati incrementalmente: se nuovi esempi si rendono disponibili dopo che il classificatore ha già appreso un modello, questo non deve essere scartato per reiniziare da capo l'apprendimento, ma l'algoritmo aggiorna naturalmente il modello pre-esistente con le nuove informazioni.

- Essendo obbligati a vedere il dataset solo un punto alla volta, questi algoritmi sono computazionalmente più efficienti: il dataset può in linea di principio essere disponibile in memoria anche solo un punto alla volta (lasciando il resto sulle periferiche), diminuendo le richieste in spazio. Queste caratteristiche rendono gli algoritmi online una buona scelta per gestire insiemi di dati di grandi dimensioni.

Stochastic Gradient Descent

Stochastic Gradient Descent[5] (SGD) è un semplice ma efficiente approccio per modelli discriminativi di classificatori lineari. Anche se SGD è presente da molto tempo nella comunità di machine learning, ha ricevuto solo recentemente una considerabile attenzione nel contesto di apprendimento su larga scala[13]. SGD è stato applicato con successo a problemi di apprendimento automatico con dati sparsi e su grande scala, spesso incontrati in *text classification* e *natural language processing*. Essendo i dati sparsi, il classificatore scala facilmente problemi che presentano anche più di 10^5 samples e 10^5 features.
I vantaggi di Stochastic Gradient Descent sono:

- Efficienza

- Semplicità di implementazione

Gli svantaggi di Stochastic Gradient Descent includono:

- SGD è sensibile e richiede un certo numero di iperparametri
- SGD è estremamente sensibile alla normalizzazione delle features

Testiamo ora le performance di questo modello, utilizzando come samples l'intero dataset a disposizione (∼103000 samples). Eseguiremo più test, variando il sottoinsieme di features utilizzate.

Features	**Features length**	**Accuracy**	**F1**	**AUC**
ALL	358	93.53%	93.22%	**93.53%**
Q+E+H+D	274	93.49%	93.19%	93.49%
D+Q	194	92.14%	91.72%	92.14%
E+H+D	190	93.45%	93.14%	93.45%
E+D	182	93.27%	92.94%	93.27%
D+H	18	92.08%	91.57%	92.08%
D	10	91.55%	91.94%	91.54%

In grassetto, la configurazione che ottiene i risultati migliori. Si nota che è la configurazione che utilizza tutte le features ad ottenerli, ma si nota anche che al decrementare, anche ingente, di numero di features le performance non scendono altrettanto velocemente.

Testiamo ora le performance, utilizzando come samples filtrando il dataset dei profili con associati un solo priors, usando quindi come minimo due priors username (∼85300 samples). Anche qui, eseguiremo più test, variando il sottoinsieme di features utilizzate.

Features	**Features length**	**Accuracy**	**F1**	**AUC**
ALL	358	93.85%	93.60%	93.84%
E+D	182	94.04%	93.77%	**94.03%**
D+H	18	93.16%	92.54%	93.14%
D	10	92.36%	91.79%	92.34%

Notiamo che in generale l'efficienza del classificatore e in particolare l'accuratezza aumentano, con ogni sottoinsieme di features. In questo caso il risultato migliore è peró ottenuto utilizzando un sottoinsieme di features e non l'insieme completo. Notiamo comunque che anche in questo caso, il sottoinsieme limitato di features D, porta a risultati accettabili.

Decidiamo ora di analizzare l'andamento del classificatore filtrando il dataset variando il numero minimo di priors e utilizzando solamente il sottoinsieme di 10 features contenute in D. In Fig 4 riportiamo i dati analizzati, attraverso un grafo che li riassume. Usando un set ristretto di solo 10 features, il classificatore spazia da una accuratezza del 91.5% utilizzando l'intero dataset fino a una accuratezza del 100% utilizzando i profili con il massimo numero di priors username (26).

Passive Aggressive

Passive Aggressive [3], abbreviato come PA, è un altro algoritmo di apprendimento online. Il suo nome è dovuto al comportamento di reazione alla perdita descritta in 4.3.1. La funzione di perdita in PA ritorna valore 0 quando il margine di accuratezza della predizione risulta essere almeno unitario. Altrimenti equivale alla differenza tra

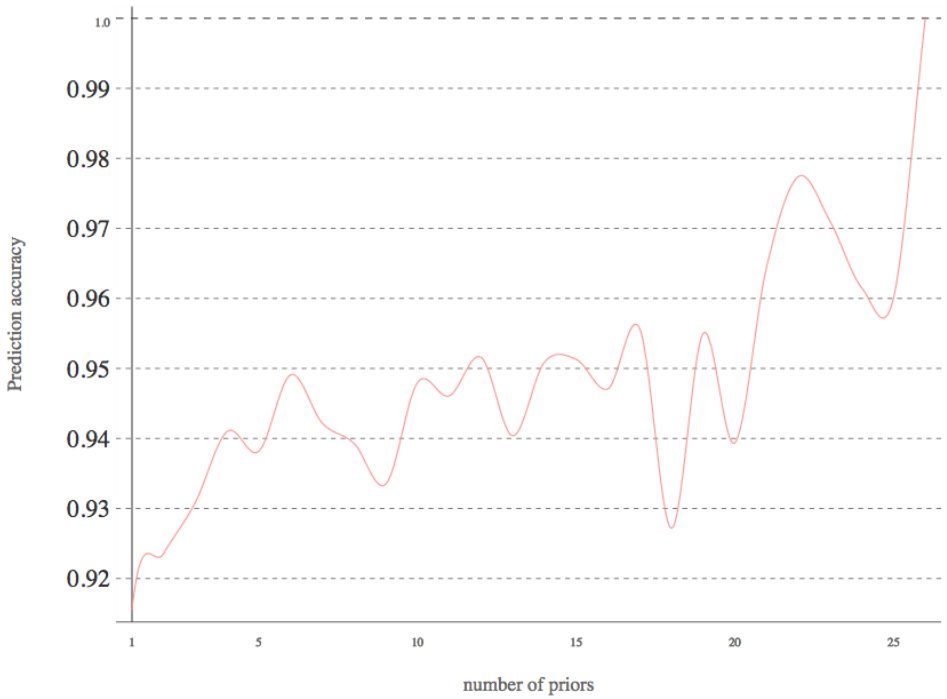


Figura 4: SGD Accuracy performance con differenti numeri di priors username

il valore di margine e 1. Il comportamento di PA, dunque, risulta passivo nel caso in cui la perdita equivalga a 0, ovvero lo stato interno del classificatore non varia e verrà restituito l'iperpiano di partenza. In caso di perdita maggiore di 0, il classificatore reagirà aggressivamente, restituendo ad ogni costo un iperpiano che soddisfi il requisito di margine quantomeno unitario. Bisogna però notare come l'aggressività nell'affrontare questo problema possa essere eccessiva in alcune situazioni reali. Infatti, è comune nella pratica avere nei dati di esempio una certa quantità di rumore sulle etichette (alcune di esse possono essere non corrette dato il vettore di esempio). Uno di questi esempi mal classificati potrebbe causare, se l'algoritmo forzasse l'ottenimento su di esso di un margine 1, la distruzione di un iperpiano che fino a lì aveva dato ottimi risultati. Per correggere ciò, gli autori introducono una versione in cui l'aggressività è calibrata da un parametro C.

Testiamo ora le performance di PA, utilizzando come samples l'intero dataset a disposizione (∼103000 samples). Eseguiremo più test, variando il sottoinsieme di features utilizzate.

Features	**Features length**	**Accuracy**	**F1**	**AUC**
ALL	358	91.44%	91.22%	91.43%
D+H	18	91.42%	90.77%	91.32%
D	10	91.66%	90.84%	**91.64%**

Generalmente con questo algoritmo si ottiene una prestazione peggiore che con SGD. Notiamo però che PA sembra essere meno sensibile alla riduzione del numero di features.

4.4 Risultati

Abbiamo testato due algoritmi di classificatori binari online, Stochastic Gradient Descent e Passive Aggressive ottenendo risultati diversi. Stochastic Gradient Descent porta generalmente a risultati migliori, ma risulta più sensibile al variare di numero di features rispetto a PA che invece mantiene un'accuratezza quasi costante al variare di queste.

Riassumendo, riteniamo Stochastic Gradient Descent l'algoritmo migliore per affrontare il problema di categorizzazione preposto in questa tesi, e individuiamo Passive Aggressive come algoritmo con risultati migliori per un numero ristretto di features.

Riporto due tabelle che riassumono i risultati appena descritti, la prima mostra le performance dei due algoritmi ottenuti utilizzando l'intero dataset e l'intero insieme features descritte nel secondo capitolo. La seconda tabella riporta le performance ottenute utilizzando un sotto insieme delle features, classificate con l'etichetta D in 4.2.3.

Model	**Accuracy**	**AUC**
Stochastic Gradient Descent	93.53%	**93.53%**
Passive Aggressive	91.34%	91.36%

Model	**Accuracy**	**AUC**
Stochastic Gradient Descent	91.55%	91.54%
Passive Aggressive	91.66%	**91.64%**

4.5 Comparazione con il lavoro svolto da Zafarani et al

Presentiamo ora alcune differenze che sussistono tra il lavoro svolto in questa tesi e quello condotto da Zafarani et al. Le differenze si distinguono in:

1. Differenze nel dataset
2. Differenze nelle features implementate

Per quanto riguarda il primo punto, vi sono più aspetti che differiscono. I due dataset differiscono per quantità, fonti, omogeneità dei SNS considerati, predisposizione dei dati ad essere classificati.

Il dataset raccolto da Zafarani et al è stato collezionato da diverse fonti[12]:

- Social Networking Sites as *Facebook* or *Google+* public data
- Blog portals as *BlogCatalog*
- Forums

Da queste fonti hanno collezionato ∼100000 coppie (c-U) dove c è uno username e U l'insieme di priors username e sia c che U appartengono allo stesso individuo. Il loro dataset conteneva username da 32 diversi siti. Le istanze negative sono state estratte da quelle positive in maniera simile a quella proposta da noi in 4.1, e bilanciate con il numero di istanze positive, raggiungendo ∼200000 istanze. Menzioniamo il fatto che il dataset raccolto nel lavoro di Zafarani risulta essere predisposto alla classificazione per le seguenti motivazioni. Su questo dataset, un sistema naive di classificazione ottiene il 77% di accuratezza. Questo sistema viene indicato nel lavoro di Zafarani come

"Exact username Match, a baseline method for comparison" e opera come descritto qui sotto.

***em*: 'Exact username Match - baseline method for comparison'** Considera un istanza positiva se lo username candidato corrisponde esattamente("*exact match*") per il α% degli username. Per impostare α accuratamente, si computa la percentuale di prior username che corrisponde esattamente al candidate username per ognuna delle istanze positive del dataset e ne viene calcolata la media secondo tutte le istanze positive.

Per il dataset di Zafarani, α viene riportato con un valore di $\sim$54%. Per meglio analizzarne l'impatto, riportano, impostano $50\% \leq \alpha \leq 100\%$. Nel migliore dei casi, con $\alpha = 50\%$, il classificatore raggiungerebbe un'accuratezza del 77%.
Sul nostro dataset, *em* con $\alpha = 50\%$, avrebbe un accuratezza del 59%. Nel nostro caso comunque, α pesato, calcolato come descritto precedentemente, assume un valore di $\sim$ 21%. Con questo valore, *em* sul nostro dataset raggiunge un'accuratezza del $\sim$ 67%. Inoltre, il dataset di Zafarani, presenta un maggior numero di profili. Riassumo le differenze dei due dataset in una tabella.

	samples	**different SNS**	α	***em***	***em*** α **= 50%**
Zafarani et al	$\sim$200000	32	$\sim$54%	<77%	77%
Questa tesi	$\sim$100000	168	$\sim$21%	67%	59%

Una seconda discrepanza con il lavoro di Zafarani si può osservare nel numero di

features implementato. Non tutte le features presentate[12] sono state infatti implementate. Altre invece, sono state aggiunte e proposte da noi. In somma, l'insieme completo delle features proposte da Zafarani conta 414 features, quelle proposte in questa tesi 358. In particolare, in questa tesi abbiamo testato le performance del classificatore utilizzando un insieme ristretto di features (10), che corrisponde a quelle introdotte da noi e che non compaiono tra quelle implementate da Zafarani. Queste sono le features etichettate come "di distanza" nel capitolo 2 e indicate con l'etichetta "D" in 4.2.3.

Ad ogni modo, anche Zafarani ha condotto un test limitando a 10 il numero di features, utilizzando *odd-rations* o OR, una misurazione che descrive la forza di associazione o dipendenza tra due valori binari, che ricopre un ruolo importante nella regressione logistica, modello utilizzato da Zafarani per affrontare il problema di classificazione.

L'idea di usare il set limitato di features "D" invece, ci viene suggerito dall'analisi della distribuzione della similarità degli username condotta in 3.1.3, dove si è notato che le classi positive e le classi negative mostravano un pattern visibilmente diverso.

	technique	**features**	**AUC**	**Accuracy**
Zafarani et al	Logistic Regression	ALL(414)	0.95	93.80%
Questa tesi	SGD	ALL(358)	0.93	93.53%
Zafarani et al	Logistic Regression	10	N/A	92.72%
Questa tesi	Passive Aggressive	10	91.46%	91.64%

Capitolo 5

Conclusioni

5.1 Lavoro svolto

In questa tesi abbiamo cercato, attraverso dati sperimentali, di provare l'esistenza di una correlazione tra username di SNS differenti che connette individui reali tra di loro e di trovare questa correlazione. Per farlo, abbiamo progettato e implementato una soluzione che consente di stabilire la correlazione di identità tra profili di individui presenti su diversi social networks online. Abbiamo collezionato un campione di $\sim$ 100000 username collegati ad altri username, di servizi differenti, per i quali è nota l'appartenenza allo stesso individuo. Per raccogliere queste informazioni abbiamo individuato una pratica e valida fonte di dati nei servizi di *social network aggregation*, che permettono di collezionare contenuti da diversi *social network services* in una presentazione unificata. Attraverso questi è possibile visualizzare tutte le attività di un utente sui diversi social network che questo ha deciso di far seguire all'aggregatore. In seguito, attraverso una tecnica per l'estrazione di dati da pagine web, conosciuta

come *web scraping* o *web data extraction*, è stata recuperata ed elaborata l'informazione di interesse, ovvero un elenco di profili di social network services appartenenti alla stessa persona. Su questi dati abbiamo cercato di individuare una serie di modelli comportamentali mostrati dagli utenti nello scegliere uno username ricercando pattern, che generando ridondanza di informazione, possono essere utili per la funzione di identificazione $f(U,c)$, così definita

$$f(U,c) = \begin{cases} 1 & \text{se } c \text{ e l'insieme } U \text{ appartengono a } I \text{ ;} \\ 0 & \text{altrimenti} \end{cases}$$

dove U è l'informazione a priori che abbiamo di un individuo I, in questo caso un insieme di username, e c è lo username candidato di cui vorremmo testare l'appartenenza allo stesso I. Questa funzione di identificazione è stata realizzata attraverso l'implementazione di un modello predittivo, in particolare un modello di riconoscimento di pattern, addestrato a riconoscere i pattern comportamentali descritti precedentemente. Questo modello è capace di attuare predizioni accurate in base alle osservazioni fatte sui dati estratti. In particolare, sono stati testati diversi modelli di apprendimento supervisionato, come Support Vector Machine (SVM) utilizzando dapprima un kernel RBF e in seguito affrontando il problema utilizzando SVM senza kernel, riscontrando performance simili, indicandoci che probabilmente il problema era risolvibile linearmente. Con il crescere del numero delle features e dei samples a disposizione, crescevano anche i tempi di esecuzione dell'algoritmo e la memoria richiesta dal calcolo, diventando troppo onerosi. È nata così la necessità di passare a un modello di classificatori diversi, conosciuti come "online", che hanno permesso di affrontare il problema con risorse di tempo e memoria molto più limitate. Tra questi abbiamo individuato e testato due algoritmi online: Stochastic Gradient Descent e Passive Aggressive.

5.2 Risultati

Abbiamo testato due algoritmi di classificatori binari online, Stochastic Gradient Descent (SGD) e Passive Aggressive (PA) usando come metriche per misurare la performance del modello di classificazione, l'Accuracy (ACC), la F-measure (F1), e la Area Under Curve (AUC) della funzione Receiver Operating Characteristic (ROC). Stochastic Gradient Descent ha portato a risultati migliori rispetto PA, raggiungendo il 93.53% di Accuracy, una F1 del 93.22% e 93.53% come AUC. Ad ogni modo, SGD risulta essere più sensibile al variare di numero di features rispetto a PA che invece mantiene un'accuratezza quasi costante al variare di queste.
Abbiamo infatti testato le performance di entrambi i classificatori variando le classi di features presentate nel capitolo due, variandone quindi il numero e riducendole fino a 10 sulle 358 implementate, ottenendo 91.55% di Accuracy e il 91.94% come F1 utilizzando SGD, e il 91.66% di Accuracy con il 90.84% come F1 utilizzando Passive Aggressive. Abbiamo inoltre analizzato l'andamento del classificatore filtrando il dataset variando il numero minimo di priors e utilizzando solamente il sottoinsieme di 10 features. Usando un set ristretto di solo 10 features, il classificatore spazia da una accuratezza del 91.5% utilizzando l'intero dataset fino a una accuratezza del 100% utilizzando i profili con il massimo numero di priors username (26). Riassumendo, riteniamo Stochastic Gradient Descent l'algoritmo migliore per affrontare il problema di categorizzazione preposto in questa tesi, e individuiamo Passive Aggressive come algoritmo con risultati migliori per un numero ristretto di features.

5.3 Sviluppi futuri

Contemporaneamente al lavoro svolto in questa tesi abbiamo individuato un'altra fonte dalla quale potere estrapolare dati riguardanti username di utenti su diversi social network. Trattasi della piattaforma per l'apprendimento di lingue **Duolingo**[1], mediante lezioni e test presentate con un modello di gamification. Il dataset ottenuto consiste in una collezione più ricca, con maggior numero di profili, e più omogenea, con un minore numero di social networks presenti. Il servizio permette ai proprio utenti di accedere utilizzando il proprio profilo Twitter, Facebook o Google+, ed è possibile visualizzare quali sono i profili associati. Nel migliore dei casi avremo quindi almeno una associazione di username tra due servizi diversi: quello selezionato per Duolingo, sempre presente, e quello del SNS utilizzato per l'accesso. La collezione è stata estratta effettuando il crawling attraverso le relazioni di following e follower presenti sul servizio, utilizzando una strategia BFS[2] e il mio profilo personale[3] come nodo di partenza, impostando un limite di profondità di ricerca a 6. Visitando nodi del grafo casualmente, e non partendo dal mio profilo, avrei probabilmente ottenuto un dataset con meno bias, ma ero interessato anche ad analizzare la mia ego-network di Duolingo. Comunque, avendo impostato un limite di profondità abbastanza alto, il numero di profili ricavato è sufficientemente alto per garantire una certa uniformità nei dati ricavati. Sono stati estratti ∼600000 profili, di cui di 350000 hanno utilizzato Facebook, 50000 hanno utilizzato Google+ e 7000 Twitter[4]. Inoltre, alcuni utenti hanno collegato più SNS, ad esempio di circa ∼21000 utenti conosciamo sia il profilo Facebook che Google+ e, ovviamente, Duolingo. Con la

[1] https://www.duolingo.com/
[2] Breadth-first search
[3] https://www.duolingo.com/mattiamattia
[4] Le cifre sono arrotondate

tecnica mostrata in 3.1.2 è possibile espandere ulteriormente il numero di coppie di username. In Fig1 riportiamo un sottoinsieme della ego-network ricavata (bolla 4), dove i colori dei nodi rappresentano la lingua di apprendimento con punteggio più alto, e la grandezza è indice sia del numero di connessioni sia del punteggio raggiunto. Una versione interattiva del grafo, filtrabile per lingua di apprendimento e indegree e con la possibilità di animare la disposizione dei nodi[5] è raggiungibile a http://mattiadmr.com/graph/duolingo/filter.html[6][7]

[5]Secondo l'algoritmo Force Atlas II
[6]Web app realizzata in HTML e Javascript
[7]Per la rappresentazione del grafo usiamo la libreria Sigma.js

Figura 1: Ego-network di Duolingo, utilizzando il mio profilo personale come nodo di partenza.

Bibliografia

[1] Facebook For Business. Audience Insights. `https://www.facebook.com/business/news/audience-insights`, 2014.

[2] Thomas M. Cover and Joy A. Thomas. *Elements of information theory*. 2012.

[3] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *The Journal of Machine Learning Research*, 7:551–585, 2006.

[4] C. Doctorow. Preliminary Analysis of LinkedIn User Passwords. `http://bit.ly/1zetKsF`, June 2012.

[5] William A. Gardner. Learning characteristics of stochastic-gradient-descent algorithms: A general study, analysis, and critique. *Signal Processing*, 6(2):113–133, 1984.

[6] George A Miller. The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological review*, 63(2):81, 1956.

[7] Daniele Perito, Claude Castelluccia, Mohamed Ali Kaafar, and Pere Manils. How unique and traceable are usernames? In *Privacy Enhancing Technologies*, pages 1–17. Springer, 2011.

[8] Wikipedia. Keyboard layouts. `http://bitly.com/kXso`.

[9] Wikipedia. Touchtyping. `http://bit.ly/1FB5GEA`.

[10] Build With. Web and Internet Technology Usage Statistics. `http://trends.builtwith.com/docinfo/OpenID`, November 2014.

[11] Reza Zafarani and Huan Liu. Connecting corresponding identities across communities. In *ICWSM*, 2009.

[12] Reza Zafarani and Huan Liu. Connecting users across social media sites: a behavioral-modeling approach. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge Discovery and Data mining*, pages 41–49. ACM, 2013.

[13] Tong Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the twenty-first international conference on Machine learning*, page 116. ACM, 2004.