Alma Mater Studiorum · Università di Bologna

Scuola di Ingegneria e Architettura
Dipartimento di Informatica · Scienza e Ingegneria · DISI
Corso di Laurea in Ingegneria Informatica

PROGETTO DI SISTEMI BASATI SU DEEP NEURAL NETWORK PER LA RILEVAZIONE DI SIMILARITÀ TRA PASSWORD

Relatore:
Prof. Marco Prandini

Presentata da:
Karina Chichifoi

Correlatore:
Davide Berardi
Andrea Melis

Anno Accademico 2019/2020

"Arguing that you don't care about the right to privacy because you have nothing to hide is no different than saying you don't care about free speech because you have nothing to say."
— Edward Snowden.

Indice

Elenco delle figure

Introduzione

La sicurezza informatica risulta importante per la salvaguardia dei dati personali. In particolare, è necessario assicurare i seguenti due aspetti:

- **Confidenzialità:** la garanzia per la quale un utente non autorizzato non possa accedere a informazione riservate.
- **Integrità:** la garanzia per la quale un utente non autorizzato non possa modificare informazioni e dati personali.

Esistono diversi meccanismi di autenticazione, divisi tra:

- "Cosa si possiede": oggetti fisici fini all'autenticazione, tra i quali chiavi, badge elettronici, token fisici, ecc.
- "Cosa si è": identificazione tramite caratteristiche fisiche, tra cui il riconoscimento facciale, l'impronta digitale, scansione dell'iride.
- "Cosa si sa": sequenze alfanumeriche che compongono una password o un PIN, informazioni personali private.

In questo documento si pone maggiore attenzione alla generazione di parole chiave, poiché risulta il meccanismo maggiormente utilizzato e facilmente raggirabile. Esistono diverse tecniche di attacco che compromettono la sicurezza che una password solitamente offre, come ad esempio gli attacchi a forza bruta che tentano tutte le possibili combinazioni di caratteri, oppure gli attacchi a dizionario che utilizzano un corpus di parole.

Le strategie di attacco possono essere di due tipologie in base a se si possiedono dati riguardanti un determinato utente: mirato, nel caso in cui si disponga di informazioni personali riguardanti l'utente e ottenute tramite ingegneria sociale o leak di dati, non mirato altrimenti.

In seguito alla creazione di molteplici profili da parte di un utente e violazioni di dati personali è spesso riscontrata la tendenza di riutilizzare la stessa password [1] o varianti di essa. Ciò può comportare la compromissione della propria identità e pericoli relativi alla privacy online.

In questo lavoro di tesi vengono analizzati i pericoli relativi al fenomeno di password reuse, e proposti approcci di prevenzione, che sfruttano tecniche di machine learning basati su lavori presenti in letteratura [2].

Si cercherà in particolare di approfondire l'approccio utilizzato, in modo da potere proporre migliori strategie fini alla analisi della similarità tra password.

1 | Stato dell'arte

La sicurezza delle password al giorno d'oggi riveste un ruolo significativo nel garantire confidenzialità e integrità dei dati personali degli utenti e delle aziende. Solitamente si è più propensi a scegliere password semplici da ricordare, come riferimenti autobiografici, oppure sequenze di caratteri molto comuni (e.g. `qwerty`, `123456`). Per semplificare la memorizzazione, si utilizzano spesso password brevi, in media da 9-10 caratteri e composte in gran parte da caratteri minuscoli [3].

Tuttavia questa scelta porta a maggiori probabilità di subire violazioni dei propri account, poiché password semplici sono vulnerabili ad attacchi di forza bruta. Inoltre, tramite tecniche di ingegneria sociale, è possibile individuare il criterio di scelta dell'utente, eventualmente ragionando sui dati disponibili grazie ai data breach.

Nel 2020 sono stati confermati 3950 data breach, dal costo medio di 3,86 milioni di dollari. Il 52% dei breach è stato causato da attacchi informatici e il numero di giorni medio per individuare un breach è stato di 207 giorni [4]. Il 42% è causato da attacchi su applicazioni web e il metodo più comune di attacco (82%) ha utilizzato credenziali rubate o ottenute tramite attacchi a forza bruta. Il 58% dei breach conteneva dati personali [5].

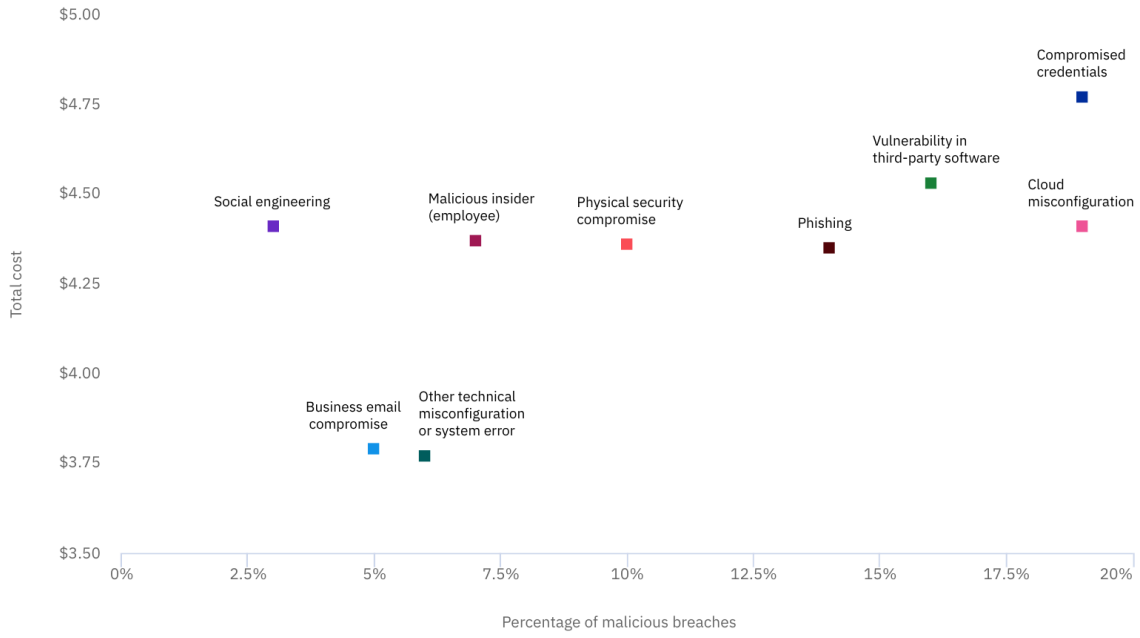

Figura 1.1: Costo medio e frequenza di data breach causati da attacchi informatici, in base alla causa, nel 2020 [4]

Sebbene la maggior parte delle password siano crittografate, è possibile risalire alla forma testuale mediante strumenti come Hashcat[1] e John The Ripper[2].

In seguito alla diffusione delle credenziali, gli utenti decidono di cambiare password e la scelta ricade spesso su varianti usate su altri account.

1.1 Scelta di una nuova password

L'utente medio ha la tendenza a scegliere password semplici [3]. Per questo motivo, spesso la nuova password è il risultato di una leggera variazione della vecchia password, o una combinazione di password precedenti [6].

Un modo per verificare la sicurezza della password è utilizzare strumenti come zxcvbn[3], che riesce a riconoscere:

- 30000 password comuni;
- nomi e cognomi comuni negli USA;
- parole spesso utilizzate in inglese su Wikipedia;
- parole spesso utilizzate alla televisione e film statunitensi;
- date;
- ripetizioni di lettere (`aaaa`);
- sequenze alfabetiche (`abcde`);
- sequenze di tastiera (`qwertyuiop`);
- il codice `l33t`.

Altri strumenti, come Kaspersky password checker[4], controllano anche dati di numerosi data breach raccolti da Have I been Pwned?[5]. Questi approcci, tuttavia, non controllano la cronologia delle password di uno specifico utente, ma soltanto la resistenza ad attacchi di forza bruta.

Per questo motivo sono state studiate strategie che tengono conto delle credenziali utilizzate. Alcune sfruttano un approccio probabilistico, come i *Bloom filter* [7]. Un'altra possibile modalità utilizza *word embedding* di password.

1.2 Word embedding

Per capire il contesto delle parole e per poterle rappresentare in base alla sfera semantica e alla sintassi, si ricorre un insieme di tecniche che prevedono il mapping delle parole o delle frasi di un dizionario in vettori di numeri reali, note come *word embedding*. Parole simili possiedono una codifica simile.

[1] `https://github.com/hashcat/hashcat`
[2] `https://github.com/openwall/john`
[3] `https://github.com/dropbox/zxcvbn`
[4] `https://password.kaspersky.com/it/`
[5] `https://haveibeenpwned.com/`

Per stabilire il valore di ogni embedding si allena una rete neurale con specifici parametri e le dimensioni variano tra 8 (per piccoli dataset) a 1024. Maggiore è la dimensione di un embedding, maggiore risulta la quantità di informazioni relativa alle relazioni tra parole [8].

1.2.1 Similarità tra parole

Per potere stabilire se due parole appartengono alla stessa sfera semantica si utilizza un metodo noto come *cosine similarity*.

$$similarity = cos(\theta) = \frac{A \cdot B}{\|A\| \cdot \|B\|} \tag{1.1}$$

Due parole risultano simili quando il valore del coseno è 1, ovvero quando l'angolo tra i due vettori risulta nullo. Si consideri il seguente esempio:

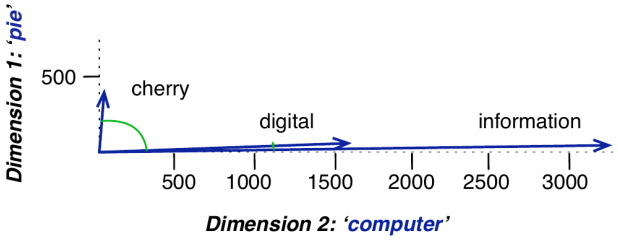

Figura 1.2: Esempio di calcolo della similarità tra word embedding [9]

Nella figura sono mostrati i vettori di 3 parole (*cherry*, *digital* e *information*) in uno spazio bidimensionale definiti dal numero di occorrenze in vicinanza alle parole *computer* e *pie*. Come si può notare, l'angolo tra *digital* e *information* risulta minore rispetto all'angolo tra *cherry* e *information*. Quando due vettori risultano più simili tra loro, il valore del coseno risulta maggiore, ma l'angolo risulta minore. Il coseno assume valore massimo 1 quando l'angolo tra i due vettori risulta nullo (0°); il coseno degli altri angoli risulta inferiore a 1 [9].

1.2.2 Tipologie di word embedding

Word2Vec

Word2Vec è un insieme di modelli architetturali e di ottimizzazione utilizzati per imparare word embedding da un vasto corpus di dati, sfruttando reti neurali.
Un modello allenato con Word2Vec riesce a individuare le parole simili tra loro, in base al contesto, grazie alla *cosine similarity* esaminata precededentemente.

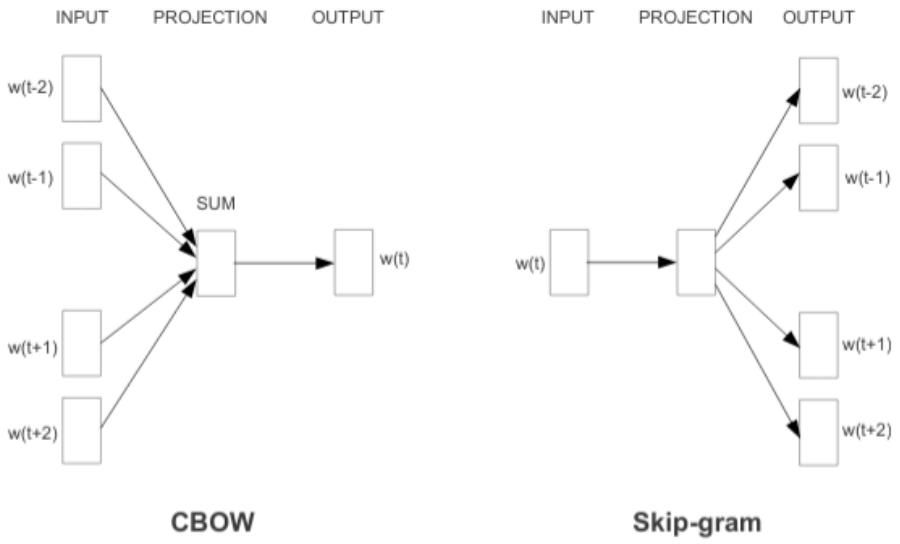

Figura 1.3: CBOW vs skip-gram [10]

Word2Vec utilizza due modelli di architetture:

- **CBOW** (continuous bag of words): l'obiettivo del training è combinare le rappresentazioni delle parole limitrofe per prevedere la parola centrale.
- **Skip-gram**: simile a CBOW, con la differenza che viene utilizzata la parola centrale per prevedere le parole circostanti relative allo stesso contesto[11].

CBOW risulta più veloce ed efficace in caso di dataset di grandi dimensioni, tuttavia, nonostante la maggiore complessità, Skip-gram è in grado di trovare parole non presenti nel corpus, per dataset di minori dimensioni [10].

FastText

FastText è una libreria open-source proposta da Facebook che estende Word2Vec, e consente un apprendimento efficiente di rappresentazioni di parole e di classificazioni di frasi. Anziché allenare un modello fornendo ogni singola parola di un dataset, FastText prevede l'apprendimento tramite *n-gram* di ciascuna parola.

Si definiscono n-gram di una parola costituita da $c_1 ... c_m$ caratteri la seguente sequenza:

$$\{c_{i_1}, c_{i_2}, \ldots, c_{i_n} \mid \sum_{j=1}^{n} i_j - i_{j-1} < 0\}$$

Ad esempio, gli n-gram della parola ciao, con $n\_mingram = 1$ e $n\_maxgram = 4$, sono i seguenti:

$$ciao = \{\{c, i, a, o\}, \{ci, ia, ao\}, \{cia, iao\}, \{ciao\}\}$$

ciao viene espresso come l'insieme di tutte le sottostringhe di lunghezza minima pari a 1, e lunghezza massima pari a 4.

FastText consente di ottenere, con più probabilità rispetto a Word2Vec, parole *out-of-dictionary*, ovvero parole sconosciute al modello in fase di training.

2 | Analisi progettuale

In questo progetto si cerca di sviluppare un sistema per verificare in modo robusto la similarità tra password di un utente, in modo da evitare di utilizzare varianti di password precedenti, e garantire maggiore sicurezza di uno o più account. A tal proposito sono stati condotti esperimenti che, per controllare la similarità tra password, utilizzavano word embedding di parole. In questo lavoro di tesi è stato preso come riferimento il paper di Bijeeta et alii [2].

2.1 Modelli di similarità tra password basati sulle reti neurali

Solitamente le metriche che misurano la robustezza di password non tengono conto della cronologia delle password passate di un utente. Ciò può costituire un problema sia se si effettua un attacco contro un utente, sia per proteggerlo:

- Durante un attacco, si considerano determinate password note grazie a leak, tuttavia non è presente un approccio flessibile che elabori varianti di password di un determinato utente.
- Non è presente un meccanismo che avverta l'utente del potenziale pericolo causato dal riutilizzo di password.

Per questo motivo, nel paper riferimento di Bijeeta et alii [2] sono stati utilizzate due tipologie di reti neurali:

- Per l'attacco è stato sviluppato un modello di rete neurale ricorrente, noto come pass2path.
- Per la difesa sono stati utilizzate tecniche di Natural Language Processing, in particolare modelli di word embedding, i quali generano una corrispondenza tra parole e vettori, mantenendo proprietà semantiche della password originale, come la similarità.

2.2 Prerequisiti

Per la creazione dei due modelli visti precedentemente, è stato utilizzato un leak disponibile sul Deep Web, di dimensione pari a 45 GB e contenente 1.4 miliardi di coppie mail-password appartenenti ad account su social come LinkedIn, MySpace, Badoo, Yahoo, Zoosk; successivamente è stato filtrato nel seguente modo:

- sono state rimosse le stringhe che contenevano 20 o più caratteri esadecimali;
- sono stati rimossi hash non decodificati;
- sono state rimosse le password più lunghe di 50 caratteri o più corte di 3 che contenevano caratteri non ASCII;
- sono stati rimossi 4528 utenti associati a centinaia di password, poiché è molto improbabile che siano account veri.

Dal risultato del processo di filtraggio sono state osservati i seguenti punti:

- la password 123456 è stata utilizzata dal 0.9% degli utenti;
- più dell'88% di password hanno una lunghezza compresa tra 6 e 12 caratteri;
- l'80% delle password contiene solo caratteri minuscoli.

Property	Values	% of PWs
Length	$3-5$	2
	$6-8$	48
	$9-12$	40
	$13-50$	10
Composition	Lower case only	80
	Upper case only	3
	Letters only	38
	Digits only	8
	Special characters only	< 0.1
	Letters & digits only	55
	Containing at least one letter, one digit and one special char	5

Figura 2.1: La distribuzione della lunghezza delle password e della loro composizione dopo la pulizia del dataset [2]

2.3 Strategie di attacco: Pass2Path

2.3.1 Euristiche e appartentenza di password

Una volta ottenuto il dataset, è necessario capire a quali persone appartengono gli account. A tal proposito vengono proposte tre euristiche:

- **Basata su email:** gli utenti vengono identificati solo dalla email. Ciascuna email appartiene a un solo utente.
- **Basata sugli username:** si considera la stringa che precede `@` nell'indirizzo email. Un utente può possiedere più mail, tutte identificate dalla stringa che precede `@`.

- **Basata su un metodo misto:** considera sia gli username che le email. Due email sono considerate connesse tra loro se hanno almeno una password in comune e se le mail connesse tra loro sono associate allo stesso username.

Per potere effettuare migliori test sul dataset sono stati rimossi gli utenti che avevano meno di due password associate alla stesso account.

Il dataset successivamente viene diviso in due parti: training set e test set (in rapporto 80%-20%). Nel paper la fase di training utilizza soltanto il dataset relativo all'euristica basata sulle email, mentre per la fase di testing si utilizza sia l'euristica basata sulle email sia quella mista.

2.3.2 Introduzione a Pass2path

Pass2path è un modello di rete neurale che consente di creare password in grado di compromettere più del 48% degli utenti in meno di mille tentativi. Il successo di questo modello è dovuto al riutilizzo della stessa password o di varianti di password da parte dell'utente. Per svilupparlo si è tenuto conto della sequenza di trasformazioni $\tau_1 ... \tau_n$, nota come *percorso* che consente, a partire dalla password $\tilde{w}$, di produrre la nuova password w.

Le modifiche sono rappresentate da una unità di misura $\tau \in \mathcal{T}$, che specifica in che posizione e quale tipo di variazione applicare in una password.
τ è composto da una tripletta $\{e, c', l\}$:

- e rappresenta una modifica da apportare;
- c' rappresenta un carattere o una stringa vuota;
- l rappresenta la posizione della modifica della password.

Le modifiche vengono classificate in tre tipologie:

- `sub` (sostituzione);
- `ins` (inserimento);
- `del` (cancellazione).

Se si considera c' sono presenti due situazioni:

- `ins` e `sub`: c' rappresenta il carattere o la stringa vuota;
- `del`: è sempre una stringa vuota.

Per ricavare il percorso tra due password, si considera quello più corto, nel seguente ordine decrescente di preferenza:

1. `del`;
2. `ins`;
3. `sub`.

Le varie trasformazioni del percorso vengono ordinate in base alla posizione della modifica.

Ad esempio, il percorso da `cats` a `kates` (distanza di modifica pari a 2) è:

$$path = \{(sub, k', 0), (ins, e', 3)\}$$

2.3.3 Come allenare pass2path

Prima di allenare pass2path è necessario tradurre ciascuna password come sequenza di tasti premuti su una *tastiera americana ANSI*. La codifica dei tasti premuti è la seguente:

- Per rappresentare una sola lettera maiuscola all'interno di una parola si pone `<s>` (che rappresenta la pressione del tasto `SHIFT`) prima della lettera, che viene lasciata minuscola. Ad esempio `Ciao` viene tradotto come `<s>ciao`.

- se si hanno più lettere maiuscole consecutive seguite da lettere minuscole si pone `<c>` (che rappresenta la pressione del tasto `CAPS LOCK`) prima e dopo la sequenza di lettere maiuscole, che viene lasciata minuscola. Per esempio `PASSword` viene tradotto come `<c>pass<c>word`.

- Nel caso di una sequenza di lettere maiuscole che si conclude alla fine della parola basta porre `<c>` all'inizio della sequenza. Per esempio `PASSWORD` viene tradotto come `<c>password` e `passWORD` come `pass<c>word`.

- Se si hanno caratteri speciali ASCII 128 si pone `<s>` davanti al carattere. Quest'ultimo viene tradotto come il tasto che viene premuto insieme a `SHIFT`. Per esempio `PASSWORD!` viene rappresentata come `<c>password<s>1`, dato che `1` viene premuto insieme a `SHIFT` e `Hello@!!` viene tradotto come `<s>hello<s>2<s>1<s>1`.

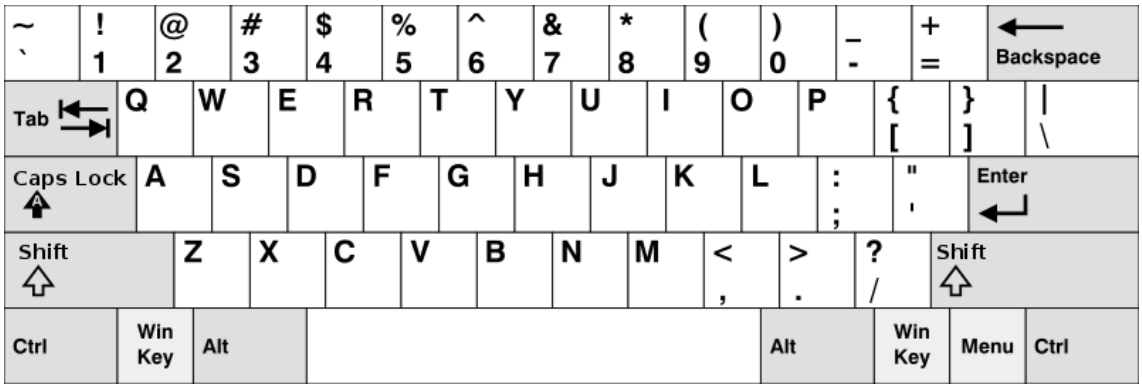

Figura 2.2: Layout della tastiera americana ANSI, fonte: Wikipedia

Successivamente occorre trovare il percorso di transizioni $\tau_1 ... \tau_n$ che consentano di trasformare la password $\tilde{w}$ in w.

Si è deciso di filtrare le password in base alla lunghezza del percorso, che deve tenere conto anche della modifica di una password basata sulla sequenza di tasti premuti. Sono state eliminate le password con un percorso di lunghezza superiore a δ.

Per allenare pass2path si utilizza il dataset di training basato sulle email. Si utilizzano due reti neurali ricorrenti (RNN) per costruire un auto-encoder [12].

2.3.4 Efficacia d'attacco con configurazioni non ripetute

Per verificare l'efficacia di attacco della rete si utilizza il dataset di test delle email, con password da indovinare w diverse dalla originale $\tilde{w}$. Con configurazioni non ripetute, Pass2path riesce in meno di 100 stime a ricavare il 13% delle password, impiegando 4 ore in tutto (Intel i9 e 128 GB di RAM, su un singolo thread).

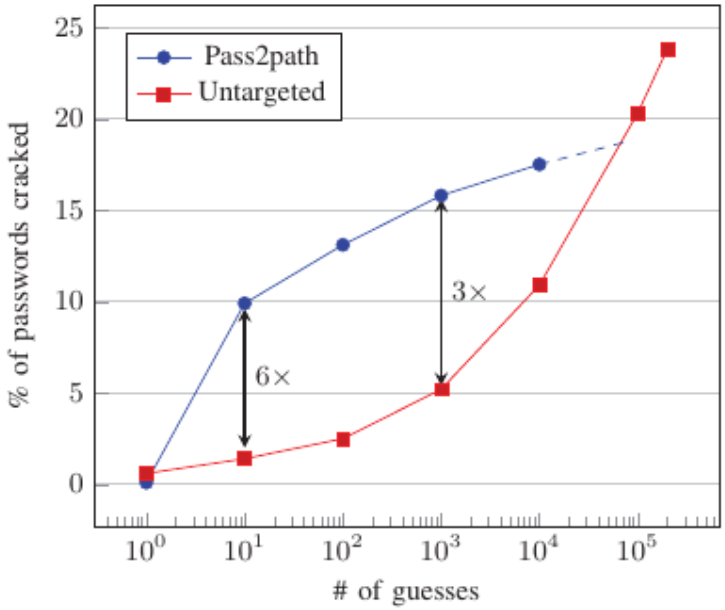

Figura 2.3: Pass2path riesce a indovinare in meno di mille tentativi una percentuale più alta di password, rispetto ad attacchi non mirati. Questi ultimi risultano efficaci soltanto se sono richiesti più di 1000 tentativi [2]

2.3.5 Efficacia d'attacco con configurazioni ripetute

Per verificare l'efficacia di un attacco della rete, in caso di password da indovinare w simile alla originale $\tilde{w}$, si utilizza il test set del dataset misto, in cui si è osservato che il 40% delle password vengono riutilizzate dagli utenti, rendendoli facili bersagli.

Come primo attacco viene utilizzata la password $\tilde{w}$, in modo da verificare il suo riutilizzo senza varianti, mentre i restanti $q-1$ vengono svolti in accordo alla tecnica di attacco scelta. Confrontando con altri modelli utilizzati, il migliore risultato è stato ottenuto da pass2path, compromettendo metà degli utenti (48.3%) in al massimo mille tentativi. Si è osservato che con password riutilizzate, aumentano le probabilità di indovinare la password con pass2path. Inoltre è importante sottolineare che gli attacchi di tipo non mirato (untargeted) ottengono migliori performance se vengono eseguiti più di 1000 tentativi; al contrario pass2path è il migliore approccio nel caso in cui si proceda con un numero minore.

I ricercatori hanno anche eseguito un vero e proprio attacco all'interno della loro università (Cornell University), in modo da potere testare su password diverse da quelle del dataset, ovvero appartenenti a studenti e professori. Il migliore risultato è stato ottenuto da pass2path, che in meno di 1000 tentativi è riuscito a scoprire la password del 8,4% degli account [2].

2.4 Difesa

2.4.1 Difesa da attacchi a dizionario mirati

Diversi studi mostrano che cambiare password non protegge completamente un utente dagli attacchi [1] [6]. Nel paper viene illustrato il concetto di PPSM (Personalized password strength meters), che sfruttano modelli preallenati di word embedding per dimostrare la sicurezza di una password. In questo modo si possono prevenire attacchi che sfruttano varianti di password presenti in data breach.

2.4.2 Personalized password strength meters

Un PPSM può essere utilizzato con lo scopo di dare un giudizio in tempo reale all'utente sulla sicurezza delle password durante la selezione. Il funzionamento è il seguente:

- vengono considerati in input una potenziale password e un set di password associate all'utente trovate in un data leak;
- vengono utilizzati due possibili criteri come output:
  - **guess rank**, ovvero il numero di tentativi di una tipologia di attacco fatti prima di indovinare la password;
  - **percentuale di similarità**, ovvero la somiglianza tra la potenziale password e la password associata all'utente.

Un possibile modo di ottenere il guess rank è basarsi su pass2path, in modo da evitare che un utente usi una password simile a quella trovata in un data breach; tuttavia prevedere password risulta costoso (dato che si considera un modello di generazione di password come pass2path) e, nel caso in cui si vogliano inviare i risultati via rete, viene occupata molta banda.

Si è osservato che risulta più efficiente e meno costoso assegnare punteggi che rappresentano la sicurezza di una password rispetto ai guess rank [2].

2.4.3 Realizzazione di un PPSM

Sotto al PPSM si trova un classificatore binario C che prende in input una password candidata w e una password nota da un leak $\tilde{w}$ e restituisce 0 se w è indovinabile in meno di 1000 tentativi a partire da $\tilde{w}$, 1 altrimenti. Per costruire tale classificatore è necessario utilizzare tecniche di word-embedding.

2.4.4 Similarità tra password via word embedding

Si definisce word embedding la tecnica di mappatura di un insieme di parole in uno spazio vettoriale d-dimensionale (solitamente con d che vale 100 o 200 o 300).

In questo modo vengono preservate le proprietà semantiche delle parole, come ad esempio la loro similarità: ad esempio, se due parole compaiono spesso nello stesso contesto, le loro rappresentazioni vettoriali avranno una distanza ridotta.

In particolare, nel problema in esame, la tecnica di word embedding viene applicata alle password: due password risultano simili se vengono scelte spesso dallo

stesso utente. Ciò permette di stabilire quanto una password sia sicura, considerando tutte le password precedentemente scelte dall'utente, e di fornire un punteggio (ovvero la percentuale di similarità).

A tal scopo, per costruire un password embedding model, viene utilizzato FastText, che impara la similarità dividendo la parola in una collezione di contesti, definiti come piccole sequenze di parole note come *n-gram*. Le parole che appaiono spesso insieme nello stesso contesto vengono definite simili.

2.4.5 Allenamento di Fasttext

Per l'allenamento del modello di FastText vengono considerati i seguenti parametri:

- **Dimensione del vettore**: impostata a 100, poiché l'allenamento del modello così risulta più rapido rispetto al parametro di default a 300.
- **Subsampling**: ignora le password che ricorrono più frequentemente. Impostato a 10^{-3}, poiché non si vogliono password con più di 1000 occorrenze.
- **Dimensione minima degli ngram**: impostata a 1, in modo che possano costruiti embedding per password mai viste durante l'allenamento.
- **Dimensione massima degli ngram**: impostata a 4, dato che le parole presenti nel dataset hanno come minimo 4 caratteri.

2.4.6 Classificazione delle password

Per classificare le password viene utilizzato un classificatore binario che restituisce 0 se le password sono simili tra loro, superando una soglia α di similarità decisa prima della misurazione, 1 altrimenti.

Una password risulta vulnerabile se, data una password w, essa viene indovinata in meno di 1000 tentativi a partire dalla password $\tilde{w}$, utilizzando come euristica Pass2Path. Per determinare la soglia α, vengono considerati 10^5 utenti dal dataset di test delle email e per ciascun utente vengono sorteggiate due password dalla collezione di password associati a essi. Una delle due password (la scelta della password è arbitraria) viene considerata come la nuova password da indovinare w, mentre l'altra password $\tilde{w}$ rappresenta la password trovata in un leak.

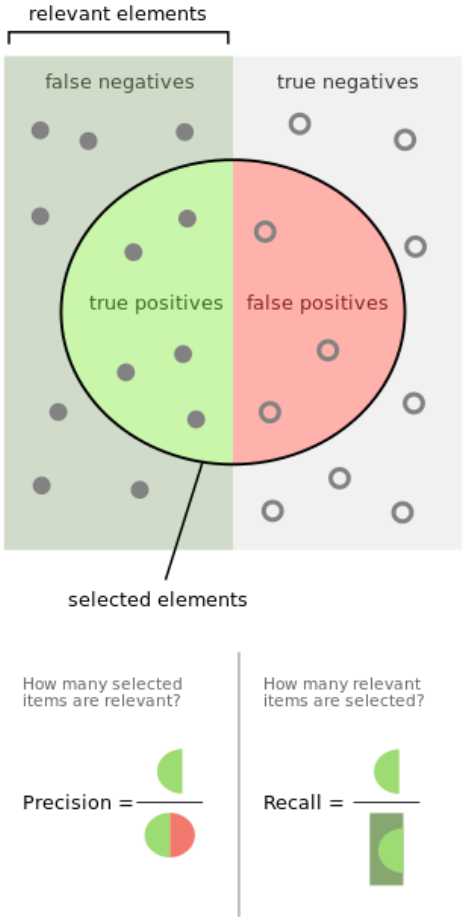

Figura 2.4: Definizione grafica di precision e recall, fonte: Wikipedia

Vengono definiti due parametri per la scelta di α:

- **Precision**: rappresenta quanti veri positivi sono stati rilevati su un totale composto da veri positivi e falsi positivi.
- **Recall**: rappresenta il numero effettivo di elementi positivi che sono stati rilevati su un totale di falsi negativi e veri positivi.

In particolare, nel caso in esame:

- per vero positivo si intende una coppia di password simili correttamente rilevate come tali;
- per falso positivo si intende una coppia di password diverse erroneamente rilevate come simili;
- per falso negativo si intende una coppia di password simili erroneamente non rilevate come tali.

Un valore di precision basso implica una imprecisa distinzione tra password simili e password non simili; un valore di recall basso invece comporta avere molte password simili non rilevate come tali.

Nel paper di riferimento di Bijeeta et alii [2] viene considerato un valore di recall molto alto (99%) e una percentuale di precision nettamente più bassa (60%). Per questo motivo α è stato posto a 0.5.

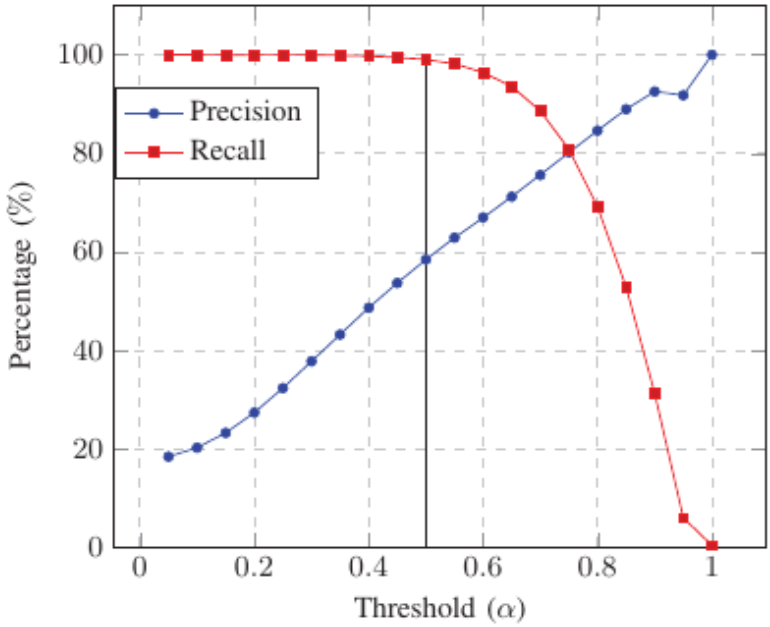

Figura 2.5: Precision e recall di Bijeeta et al. [2]

2.4.7 Modelli compressi di word embedding

Il modello prodotto da Bijeeta et alii [2] è stato successivamente compresso, in modo da ridurre la dimensione da 1.5 GB a 3 MB, senza che la qualità delle performance venisse ridotta, mediante tecniche di quantizzazione. Per la produzione del modello compresso si è tenuto conto del parametro η che determina il rapporto di compressione. Per la valutazione della compressione, sono stati prodotti più modelli in base a differenti valori η; infine è stato scelto il modello con $\eta = 5$ e dimensione complessiva di 3 MB, poiché il valore di recall non ha subito notevoli variazioni.

η	**Size (MB)**	**Precision** (%)	**Recall** (%)
100	50.0	59.1	99.3
10	5.3	48.5	99.0
5	3.0	41.3	98.6

Figura 2.6: Precision e recall in base al rapporto di compressione η del modello di Bijeeta et alii [2]

2.5 Modifiche e analisi del progetto

Per questo progetto sono state effettuate diverse scelte:

- Si è scelto di non implementare pass2path, per motivi di complessità e di costo in termini di testing;
- sono stati modificati i criteri di filtraggio del dataset per allenare il modello di word embedding;
- sono state utilizzate euristiche diverse rispetto a pass2path;
- sono stati modificati i parametri per allenare la rete neurale che ricava la similarità tra password.

3 | Implementazione

Lo scopo del progetto è sviluppare un classificatore che, date due password, determini se sono simili tra loro (e quindi potenzialmente non sicure) o meno.

A tal proposito sono stati implementati modelli di word embedding, i quali permettono di rappresentare parole in uno spazio vettoriale preservando proprietà semantiche quali la similarità e la co-occorrenza nel medesimo contesto (crf. paragrafo 1.2). Come modello è stato scelto FastText, il quale sfrutta informazioni sugli n-gram della parola per determinarne l'embedding [2].

Si è deciso di allenare 5 diversi modelli:

- Il modello di Bijeeta et alii [2] che utilizza la libreria `word2keypress` per ricavare la sequenza di tasti premuti per specifici caratteri, numero minimo di n-gram pari a 1, e numero di epoche per allenare la rete pari a 5.
- Un modello che utilizza `word2keypress`, con numero minimo di n-gram pari a 2, e numero di epoche per allenare la rete pari a 10.
- Tre modelli che non rilevano la sequenza di tasti premuti:
  - Uno con numero minimo di n-gram pari a 1, e numero di epoche per allenare la rete pari a 5.
  - Uno con numero minimo di n-gram pari a 2, e numero di epoche per allenare la rete pari a 5.
  - Uno con numero minimo di n-gram pari a 2, e numero di epoche per allenare la rete pari a 10.

3.1 Precondizioni

Prima della fase di allenamento della rete, è necessario avere un dataset valido di password; nel progetto è stato utilizzato lo stesso data breach da 45 GB citato da Bijeeta et alii [2], tuttavia sono stati scelti criteri diversi di filtraggio:

- Sono state rimossi gli account con password con lunghezza inferiore a 4 caratteri o maggiore di 30.
- Sono state rimosse gli account con password che presentavano caratteri non ASCII o non stampabili.
- Sono stati rimossi gli account creati da bot, riconoscibili grazie al numero di occorrenze della stessa email nel dataset, superiore a 100.

- Sono stati rimossi gli account con password contententi sequenze esadecimali (identificate da `$HEX[]` e `\x`).
- Sono stati rimossi sequenze che rappresentano caratteri in HTML, come:
  - `&gt` (simbolo $>$);
  - `&ge` (simbolo $\geq$);
  - `&lt` (simbolo $<$)
  - `&le` (simbolo $\leq$);
  - `&#` (ovvero i codici di entità in HTML);
  - `amp`.
- Sono stati rimossi gli account che presentavano meno di 2 password, poiché più password per utente risultano indispensabili per ricavare la similarità.

Successivamente, in accordo con Bijeeta et alii [2], per i modelli che prevedevano la memorizzazione delle sequenze di tasti premuti, è stata utilizzata la libreria python `word2keypress`[1]. Successivamente i risultati sono stati salvati in un file `csv` nel seguente formato:

```
sample@gmail.com:  ["'97314348'", "'voyager<s>1'"]
```

3.2 Impostazioni dell'ambiente

Per potere allenare il modello è stato utilizzato `gensim.FastText` [13].
`gensim` è una libreria python multipiattaforma e open source che racchiude un'ampia scelta di modelli di word embedding pre allenati [13]. Per estrapolare i dati dal file `csv` è stata sviluppata una classe ausiliaria `PasswordRetriever`.

3.3 Allenamento del modello

3.3.1 Parametri di FastText

```
1 negative = 5
2 subsampling = 1e-3
3 min_count = 10
4 min_n = 2
5 max_n = 4
6 SIZE = 200
7 sg = 1
```

Per questo progetto è stato utilizzato il modello *Skip-gram* (`sg = 1`) e il negative sampling [14]:

[1] `https://github.com/rchatterjee/word2keypress`

- Il modello skip-gram (parametro `sg = 1`) è stato scelto in quanto la rappresentazione distribuita dell'input è stata utilizzata per prevedere il contesto delle password. In particolare, risulta efficace per determinare quali caratteri intorno a una specifica sequenza sono presenti; in questo modo il modello riesce a imparare password non presenti nel corpus fornito per l'allenamento. [11]

- Il negative sampling (parametro `negative = 5`) rende l'allenamento più veloce, poiché ciascuna sezione dell'allenamento aggiorna solo una piccola percentuale dei pesi del modello. [14] Per dataset di dimensioni maggiori (come in questo caso) è consigliabile impostare il suo valore tra 2 e 5. [13]

- La dimensione del vettore contenente gli embedding è impostato a 200 (parametro `SIZE = 200`), in modo da potere allenare più velocemente il modello. Normalmente viene raccomandata una dimensione pari a `SIZE = 300`. [13]

- Il subsampling ignora le password più frequenti, ovvero che presentano più di 1000 occorrenze.

- `mincount` rappresenta il numero minimo di occorrenze di una password nel dataset di training affinché venga considerata nell'allenamento [2] [13].

- `min_n` e `max_n` rappresentano rispettivamente il numero minimo e massimo degli n-gram. Gli n-gram vengono utilizzati per prevedere una sequenza di caratteri e il contesto di quest'ultima. In questo caso essi rappresentano una sequenza di caratteri contigui e il loro scopo è di dare informazioni di contesto e posizionali di una determinata sequenza all'interno di una password [2] [13].

3.3.2 Allenare FastText

La lista di password relativa a ciascun utente (`password_list`) viene ottenuta grazie alla classe ausiliaria `PasswordRetriever`. Il modello di FastText si basa su `password_list` e viene allenato con i parametri elencati precedentemente.

```
1 filename='../train.csv'
2 password_list = PasswordRetriever(filename)
3 trained_model = FastText(password_list, size=SIZE, min_count=min_count,
4                          workers=12, negative=negative,
5                          sample=subsampling, window=20,
6                          min_n=min_n, max_n=max_n)
```

3.4 Compressione del modello

Il modello allenato ha un peso complessivo pari a 4.8 GB, e ciò comporta i seguenti problemi:

- Meno efficiente in termini di spazio, di conseguenza l'utilizzo del modello è limitato su sistemi con vincoli di memoria o di spazio.

- È difficile utilizzare il modello in contesti distribuiti, poiché non è facilmente trasportabile.

Per comprimere il modello si è utilizzata la libreria `compress_fasttext`, che sfrutta tecniche di quantizzazione e di feature selection. [15]

```
1 big_model = gensim.models.fasttext.load_facebook_vectors('model.bin')
2 small_model = compress_fasttext.prune_ft_freq(big_model, pq=True)
3 small_model.save('compressed_model')
```

Viene definito come *feature selection* il processo di selezione delle feature più importanti da usare per costruire un modello [16] [17].

Per *Product quantization* si intende un particolare tipo di quantizzazione vettoriale, che viene utilizzata per la compressione di modelli di linguaggio naturale e di elaborazione di immagini e consente di generare in modo non esponenziale una quantità grande di codice in tempi contenuti e con costi ridotti in termini di memoria [2] [15] [18].

Il modello compresso ottenuto dalle operazioni di quantizzazione e di feature selection ha una dimensione di 20 MB.

4 | Risultati

4.1 Differenze tra modello normale e compresso

Dati i problemi che comportavano l'utilizzo dei modelli da 4.8 GB, si è scelto di utilizzare per l'analisi dei risultati le versioni compresse da appena 20 MB ottenute tramite product quantization. Per misurare la differenza di prestazioni tra il modello originale e la sua versione compressa si è considerato come riferimento il modello di Bijeeta et alii, avente le seguenti caratteristiche:

- traduzione della sequenza dei tasti premuti con `word2keypress`
- numero minimo di n-gram pari a 1;
- numero di epoche di training pari a 5.

Per entrambi i modelli si è tenuto conto del valore di precision e recall, in modo da fornire una valutazione efficace del modello. Non sono state osservate differenze significative riguardanti i valori, motivo per il quale si è scelto di considerare soltanto le versioni compresse per valutare gli altri modelli.

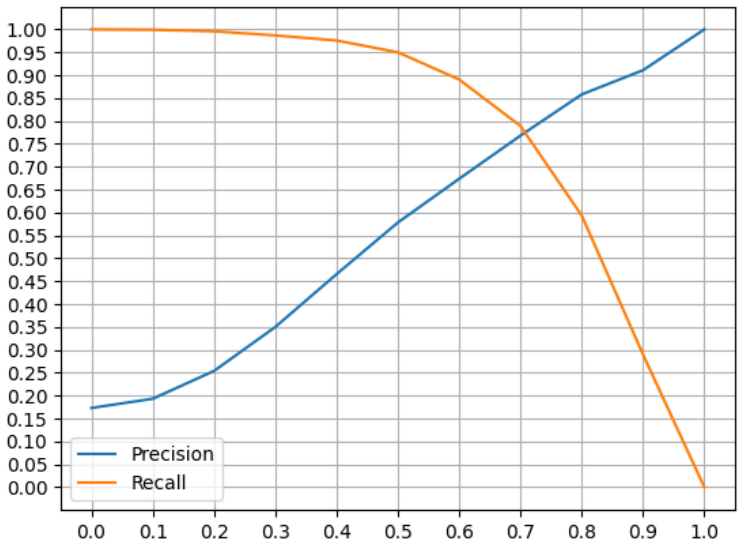

Figura 4.1: Precision e recall nel modello non compresso di Bijeeta et al. [2] con le euristiche definite nel paragrafo 4.2.1, con `word2keypress`, `n_mingram = 1`, `epoche = 5`

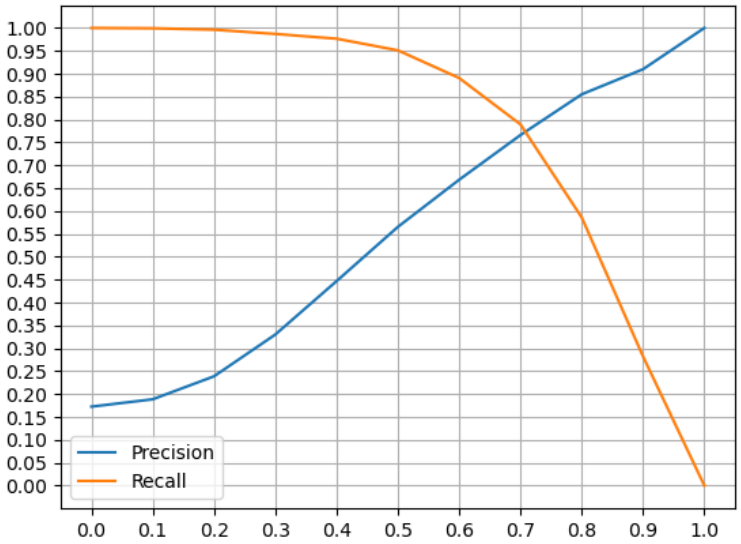

Figura 4.2: Precision e recall nel modello compresso di Bijeeta et al. [2] con le euristiche definite nel paragrafo 4.2.1, con `word2keypress`, `n_mingram = 1`, `epoche = 5`

4.2 Classificazione del modello

4.2.1 Euristiche adottate

Per valutare i modelli, a differenza di Bijeeta et alii [2], non si è utilizzato Pass2Path a causa della complessa implementazione. Si è invece adottata la seguente euristica:

- verifica della password con la variante minuscola;
- verifica della password con la variante maiuscola;
- verifica della password con la traduzione in codice `l33t`;
- verifica se l'edit distance della password supera 5.

4.2.2 Ground truth e prediction

Per *ground truth* si intende il risultato ideale che ci si aspetta e viene utilizzato per verificare una correttezza delle previsioni del modello. [19]

Il termine *prediction* si riferisce all'output di un modello dopo che è stato allenato su un dataset e applicato a nuovi dati quando si vuole prevedere la probabilità di un certo evento. [20].

Per calcolare il valore di ground truth si tiene conto delle candidate due password e della euristica scelta; per ottenere il valore di prediction invece si considera si utilizza la funzione `gensim.similarity` del modello allenato, che sfrutta la cosine similarity discussa nel capitolo 1.2.1. Nel caso in cui si consideri la variante con `word2keypress` occorre convertire le due password in sequenza di tasti premuti prima di ricavare il valore di prediction.

4.2.3 Precision e recall

Per ricavare precision e recall si utilizza l'approccio di cui accennato nel paragrafo 2.4.6; a questo scopo si definiscono i seguenti parametri:

- **Veri positivi (TP)**: viene incrementato se sia la prediction che la ground truth hanno valore positivo non nullo;
- **Falsi positivi (FP)**: viene incrementato se la ground truth è nulla e la similarità è positiva non nulla;
- **Falsi negativi (FN)**: viene incrementato se la ground truth è positiva non nulla e la similarità è nulla;

Successivamente si ricava il valore di precision e recall nel seguente modo (cfr. paragrafo 2.4.6):

$$precision = \frac{TP}{TP + FP}$$
$$recall = \frac{TP}{TP + FN}$$

4.3 Similarità: un confronto

In base a quanto spiegato nel paragrafo 2.4.6 si sono ottenuti i seguenti grafici:

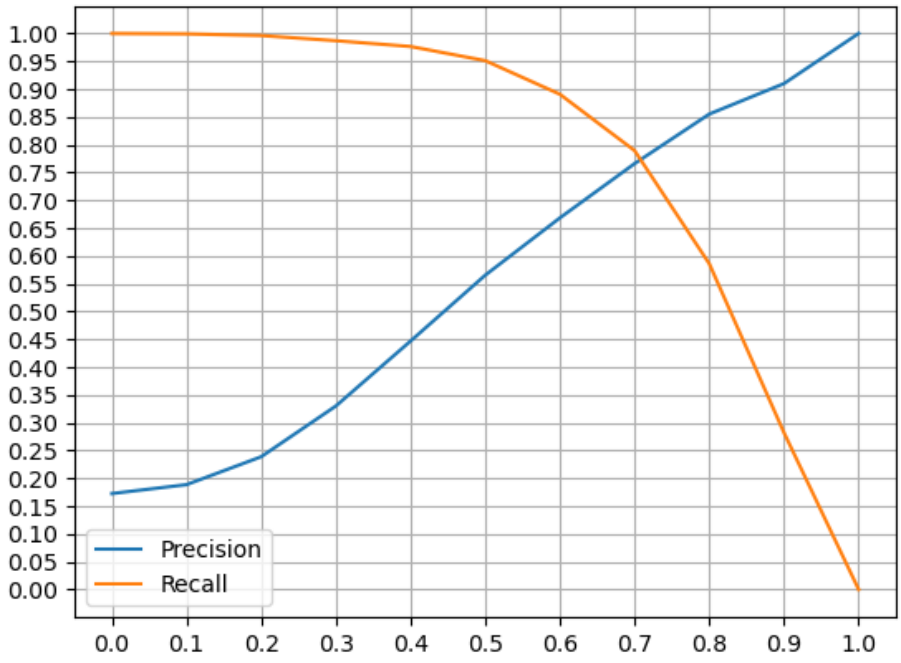

Figura 4.3: Precision e recall nel modello di Bijeeta et al. [2] con le euristiche definite nel paragrafo 4.2.1, con `word2keypress`, `n_mingram = 1`, `epoche = 5`

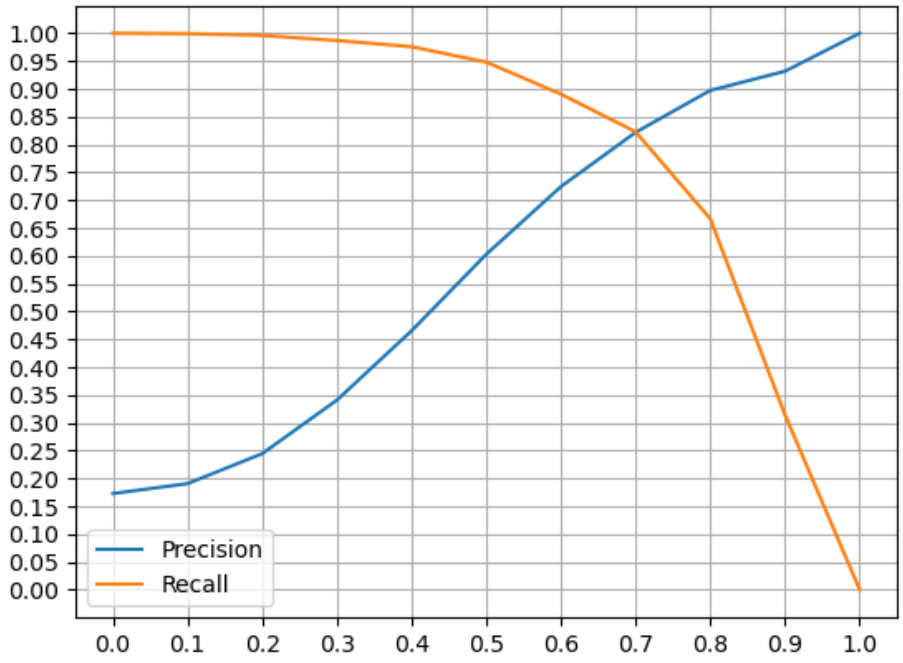

Figura 4.4: Precision e recall nel modello senza `word2keypress`, `n_mingram = 1`, `epoche = 5`

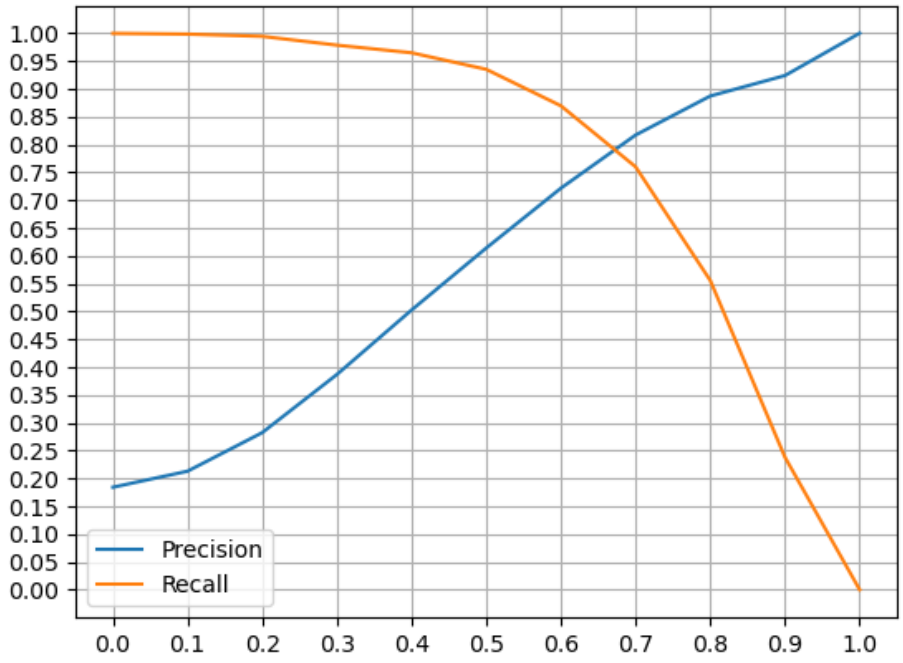

Figura 4.5: Precision e recall nel modello con `word2keypress`, `n_mingram = 2`, `epoche = 10`

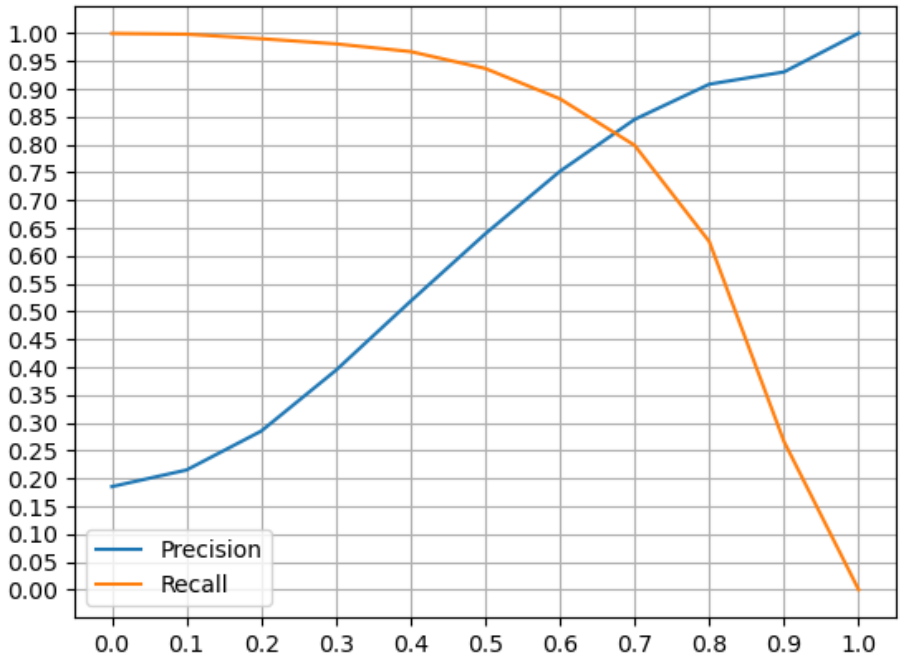

Figura 4.6: Precision e recall nel modello senza `word2keypress`, `n_mingram = 2`, `epoche = 10`

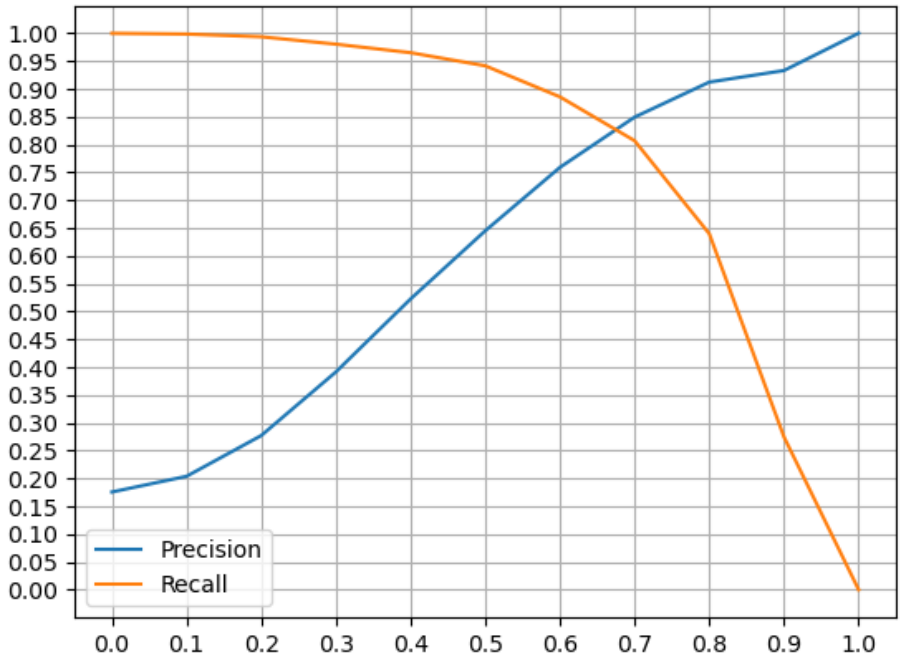

Figura 4.7: Precision e recall nel modello senza `word2keypress`, `n_mingram = 2`, `epoche = 5`

4.4 Risultati ottenuti

Come accennato in 2.4.6, per classificare le password è necessario definire una soglia α di similarità, in modo da definire due password simili tra loro se la loro similarità supera α, diverse tra loro altrimenti. Analizzando i modelli ottenuti, si possono osservare i seguenti risultati:

- con `word2keypress`, `n_mingram = 1`, `epoche = 5` (modello di Bijeeta et alii [2], figura 4.3):
  - $\alpha = 0.5$: precision pari a $\simeq 57\%$ e recall pari a $\simeq 95\%$
  - $\alpha = 0.6$: precision pari a $\simeq 67\%$ e recall pari a $\simeq 89\%$
- senza `word2keypress`, `n_mingram = 1`, `epoche = 5` (figura 4.4):
  - $\alpha = 0.5$: precision pari a $\simeq 60\%$ e recall pari a $\simeq 95\%$
  - $\alpha = 0.6$: precision pari a $\simeq 73\%$ e recall pari a $\simeq 88\%$
- con `word2keypress`, `n_mingram = 2`, `epoche = 10` (figura 4.5):
  - $\alpha = 0.5$: precision pari a $\simeq 62\%$ e recall pari a $\simeq 94\%$
  - $\alpha = 0.6$: precision pari a $\simeq 73\%$ e recall pari a $\simeq 87\%$
- senza `word2keypress`, `n_mingram = 2`, `epoche = 10` (figura 4.6):
  - $\alpha = 0.5$: precision pari a $\simeq 65\%$ e recall pari a $\simeq 94\%$
  - $\alpha = 0.6$: precision pari a $\simeq 75\%$ e recall pari a $\simeq 88\%$
- senza `word2keypress`, `n_mingram = 2`, `epoche = 5` (figura 4.7):
  - $\alpha = 0.5$: precision pari a $\simeq 65\%$ e recall pari a $\simeq 95\%$
  - $\alpha = 0.6$: precision pari a $\simeq 77\%$ e recall pari a $\simeq 89\%$

I modelli con i migliori risultati non utilizzano la libreria `word2keypress`, e presentano un miglioramento di almeno il 3% nel valore di precision e con valore di recall invariato, rispetto ai modelli che utilizzano `word2keypress`.

Un altro importante miglioramento è stato determinato dal valore di `n_mingram`: tutti i modelli con `n_mingram = 2` hanno avuto un lieve incremento del valore di precision.

Il modello con i risultati più scarsi è quello con `word2keypress`, `n_mingram = 1`, `epoche = 5` (modello di Bijeeta et alii [2], figura 4.3), e lo si può osservare dai valori inferiori di precision rispetto ad altri modelli.

Il modello con le migliori performance è invece riportato in figura 4.7, senza `word2keypress`, `n_mingram = 2`, `epoche = 5`, con valore di precision pari a 77%, per $\alpha = 0.6$, superiore di $\simeq 10\%$ rispetto al modello di Bijeeta et alii [2].

4.4.1 Criticità del modello proposto da Bijeeta et alii

I motivi per cui il modello di Bijeeta et alii [2] è risultato il peggiore sono i seguenti:

- La libreria `word2keypress` traduce ciascun carattere come sequenza di tasti premuti, come accennato nel paragrafo 2.3.3. Di conseguenza, quando si hanno due password che presentano un'alternanza di caratteri minuscoli e maiuscoli, esse risulteranno molto simili tra loro, a causa della ripetizione delle sequenze `<s> <c>`.

- Sempre a causa di `word2keypress`, la presenza di caratteri speciali non consecutivi che si ottengono premendo `SHIFT`, ad esempio:

  - la password `$1mp@t1c*`, che viene tradotta come `<s>41mp<s>2t1c<s>8`)
  - la password `#wlng%p*m}` che viene tradotta come `<s>3wlng<s>5p<s>8m<s>[`

  rende la valutazione di similarità tra due password poco affidabile, poiché verrebbero valutate come simili, quando in realtà dovrebbero essere valutate come sicure e diverse tra loro.

- L'utilizzo di `n_mingram = 1` rende la valutazione meno precisa rispetto allo stesso modello con `n_mingram = 2`, poiché risulta più efficace valutare la vicinanza di due specifici bigrammi, anziché avere una valutazione maggiormente dispersiva in cui si considera la posizione di un carattere rispetto a un altro. Nelle password, infatti, risulta che i singoli caratteri non dipendano da un insieme di regole (come nel caso della letteratura inglese), ma vengano fortemente influenzati da fattori distinti tra loro e difficilmente rappresentabili.

4.4.2 Soglia di similarità per la valutazione di due password

Per potere determinare se due password siano simili tra loro occorre definire la soglia di similarità α, tenendo in considerazione i valori di precision e recall ottimali.

Nel caso in studio, è necessario avere un valore di recall più alto rispetto al valore di precision, dato che è importante rilevare più password simili tra loro possibili. Questo tuttavia, per valori di precision molti bassi (come accennato nel paragrafo 2.4.6) può comportare un'alta percentuale di alti positivi, ovvero di coppie di password considerate come simili, anche se abbastanza diverse tra loro.

Un buon compromesso è stato raggiunto con $\alpha = 0.6$ (rappresentato dalle ascisse dei grafici con precision e recall); nel modello mostrato in figura 4.7 si ha, per $\alpha = 0.6$ precision pari a $\simeq 77\%$ e recall pari a $\simeq 89\%$.

Nel paper di Bijeeta et alii [2] è stato scelto $\alpha = 0.5$, tuttavia, come si può vedere in figura 2.4, ciò comportava un valore di precision basso, con una variazione di recall pari a $\simeq 5\%$ e di precision pari a $\simeq 10\%$ rispetto ai valori registrati con $\alpha = 0.6$.

4.5 Rappresentazione grafica della distanza tra parole

In questo progetto è stato scelto, per facilitare la comprensione, di rappresentare le similarità tra password mediante un grafico a 3 dimensioni. A questo scopo è stato utilizzato un algoritmo noto come t-SNE[1] per ridurre la dimensione del modello da 200 a 3. Date due password come `ipwnedyou` e `numBerOne` si possono vedere per ciascuna di esse le 5 password più simili proposte dal modello e la distanza in termini di similarità rispetto alla password originaria.

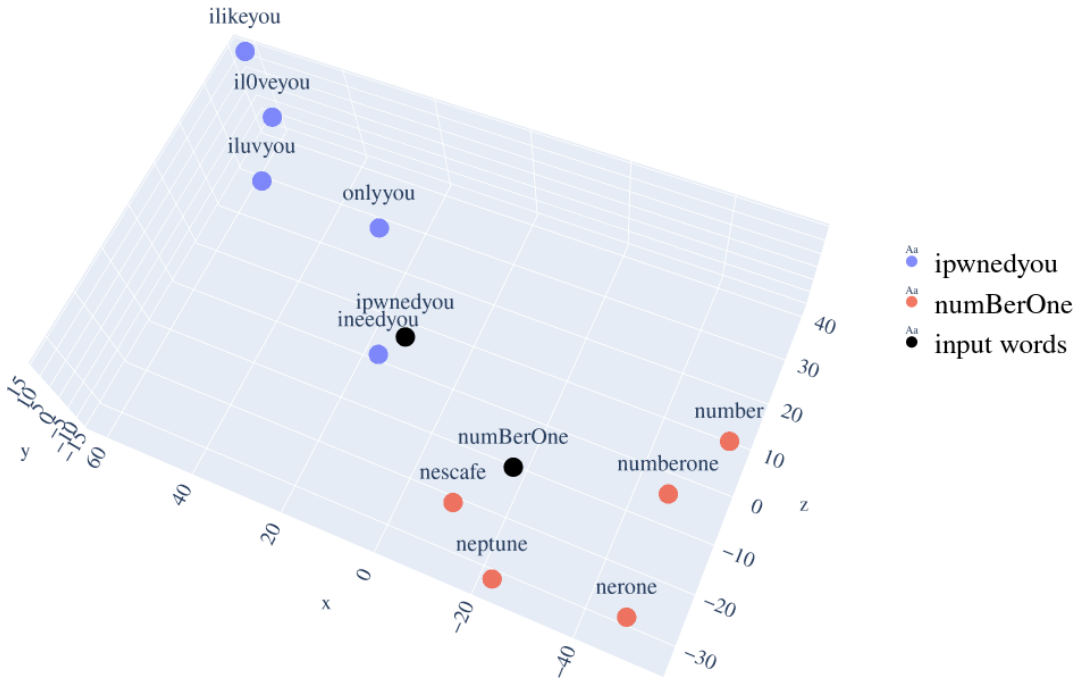

Figura 4.8: Grafico 3D per mostrare la similarità tra password

[1] https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

Conclusioni

In questo progetto di tesi ci si è soffermati sul problema dell'utilizzo di varianti di password precedenti e di possibili approcci di prevenzione. Per potere comprendere quali siano le varianti ci si è soffermati non solo sulla forma delle password, ma anche sulla sfera semantica che ricoprono, grazie a tecniche di word embedding, che consentono la traduzione di parole in vettori di numeri reali. A tal scopo è stato utilizzato `gensim.FastText` come modello di word embedding per potere fornire un punteggio di similarità tra due password, il cui apprendimento è influenzato dall'insieme di `n-gram` di una parola.

Prima dell'allenamento di FastText, è stato filtrato un dataset da 43 GB, escludendo gli utenti con meno di due password; successivamente si è diviso il dataset in training set e test set e si è deciso di allenare 5 modelli e di confrontarli tra loro, in base ai seguenti fattori:

- Conversione mediante libreria `word2keypress` delle password in sequenza di tasti premuti per ottenere caratteri maiuscoli e caratteri speciali.
- Numero minimo di n-gram, ovvero la lunghezza minima della sequenza di caratteri che compone in parte la password.
- Numero di epoche con cui è stato allenato il modello.

Dopodiché, grazie al metodo di `product quantization`, è stato compresso il modello, in modo da ridurre la dimensione da 4.8 GB a 20 MB, senza subire perdita di qualità. In seguito, è stata scelta un'euristica che, per la valutazione della similarità di due password, effettuasse i seguenti controlli:

- verifica della password con la variante maiuscola;
- verifica della password con la variante minuscola;
- verifica della password con la traduzione in codice `l33t`;
- verifica se l'edit distance della password superasse 5.

Infine è stato ricavato il grafico di precision e recall, utile per potere fornire una valutazione dei modelli e della loro affidabilità, ed è stato scoperto che il modello implementato dal paper di riferimento di Bijeeta et alii [2] che utilizza l'euristica precedentemente citata risulta il peggiore in termini di performance. Ciò era dovuto alla traduzione da parte di `word2keypress` di caratteri in sequenze composte da `<c>` `<s>`, comuni a più password che utilizzavano una alternanza di maiuscole e caratteri speciali.

Un altro problema del modello precedentemente citato era determinato dal numero minimo di n-gram, pari a 1, che risultava troppo dispersivo per potere consentire un apprendimento efficace da parte del modello dei caratteri che componevano una password.

Per risolvere, si è deciso di allenare un modello, che non tenesse conto della sequenza di tasti premuti e che, in fase di apprendimento, dalla durata di 5 epoche, considerasse un numero minimo di n-gram pari a 2.

Quest'ultimo è risultato il migliore tra tutti i modelli allenati.

4.6 Sviluppi futuri

Sono possibili diversi utilizzi del modello ottenuto in questo progetto:

- In un ambiente distribuito dedicato al cambio di password, dove l'utente possa fornire la vecchia password e la nuova. Il modello compresso può determinare se la nuova password possa essere considerata abbastanza sicura rispetto a quella vecchia, e può essere inviato come payload JavaScript, mentre il controllo di sicurezza può essere effettuato lato client, assicurandosi che la nuova candidata password non venga inoltrata a un server remoto, prima di diventare definitiva. Un vantaggio nell'utilizzare i modelli compressi risulta nel garantire l'anonimizzazione delle password, poiché esse vengono rappresentate come word embedding, e non sono reversibili, pertanto in fase di intercettazione non comportano alcun rischio. Successivamente il server può effettuare un controllo sul database per vedere se la password risulta sicura rispetto alla versione precedente. Questo comporta non dovere salvare in un database le password in chiaro e non dovere ricorrere ad algoritmi di crittografia, grazie alle proprietà dei word embedding.

- Confronto con il modello che sfrutta i bloom filter, per poi stabilire quale approccio sia il migliore.

Ringraziamenti

Ringrazio i miei relatori per il supporto e per avermi dato l'opportunità di approfondire questa tema.

Altrettanto ringrazio i miei genitori e tutti coloro che mi sono stati vicini in questo percorso e hanno creduto in me. A loro è dedicata questa tesi.

Bibliografia

[1] *Online Security Survey*. 2019. URL: https://services.google.com/fh/files/blogs/google_security_infographic.pdf.

[2] B. Pal et al. «Beyond Credential Stuffing: Password Similarity Models Using Neural Networks». In: (2019), pp. 417–434. DOI: 10.1109/SP.2019.00056.

[3] Sarah Pearman et al. «Let's Go in for a Closer Look: Observing Passwords in Their Natural Habitat». In: *Commun. ACM* 50.1 (ott. 2017), pp. 295–310. ISSN: 1557-735X. DOI: 10.1145/3133956.3133973. URL: https://dl.acm.org/doi/pdf/10.1145/3133956.3133973.

[4] *Cost of a Data Breach Report 2020*. 2020. URL: https://www.ibm.com/security/digital-assets/cost-data-breach-report/#/pdf.

[5] *Data Breach Investigations Report 2020*. 2020. URL: https://enterprise.verizon.com/resources/executivebriefs/2020-dbir-executive-brief.pdf.

[6] *Password Usage Study: How Do We Use Passwords?* 2019. URL: https://web.archive.org/web/20200610025025/https://www.hypr.com/wp-content/uploads/password_usage_study_infographic_hypr.png.

[7] Davide Berardi et al. «Password Similarity Using Probabilistic Data Structures». In: *Journal of Cybersecurity and Privacy* 1.1 (2021), pp. 78–92. ISSN: 2624-800X. DOI: 10.3390/jcp1010005. URL: https://www.mdpi.com/2624-800X/1/1/5.

[8] *Tensorflow tutorial: word embeddings*. 2021. URL: https://www.tensorflow.org/tutorials/text/word_embeddings.

[9] Daniel Jurafsky e James H. Martin. *Speech and Language Processing*. 2020. URL: https://web.stanford.edu/~jurafsky/slp3/ed3book_dec302020.pdf.

[10] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL].

[11] *Word representations: FastText tutorial*. URL: https://fasttext.cc/docs/en/unsupervised-tutorial.html.

[12] Alex Sherstinsky. «Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network». In: *Physica D: Nonlinear Phenomena* 404 (2020), p. 132306. ISSN: 0167-2789. DOI: 10.1016/j.physd.2019.132306. URL: http://dx.doi.org/10.1016/j.physd.2019.132306.

[13] *Gensim FastText model*. URL: https://radimrehurek.com/gensim/models/fasttext.html.

[14] *Word2Vec Tutorial Part 2 - Negative Sampling*. 2017. URL: https://web.archive.org/web/20210228151248/http://mccormickml.com/2017/01/11/word2vec-tutorial-part-2-negative-sampling/.

[15] *compress-fasttext*. URL: https://pypi.org/project/compress-fasttext/.

[16] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[17] *Feature Selection and Dimensionality Reduction*. 2019. URL: https://web.archive.org/web/20210228115414/https://towardsdatascience.com/feature-selection-and-dimensionality-reduction-f488d1a035de?gi=664810fe2242.

[18] Tiezheng Ge et al. «Optimized product quantization». In: *IEEE transactions on pattern analysis and machine intelligence* 36.4 (2013), pp. 744–755.

[19] Yves Lemoigne e Alessandra Caner. *Molecular Imaging: Computer Reconstruction and Practice*. Springer Science & Business Media, 2008.

[20] *DataRobot wiki: prediction*. URL: https://www.datarobot.com/wiki/prediction/.