University of Padova

Department of Information Engineering

Master Thesis in ICT for Internet and Multimedia

Privacy Leakage Analysis of Text Representations for Natural Language Processing

Supervisor
Prof. Alberto Testolin

Master Candidate
Luca Corò

Academic Year 2021/2022

Abstract

Natural language processing (NLP) offers powerful tools, such as text suggestions to help users be more efficient or classifiers that categorize documents by their content. These tools are usually powered by machine learning (ML) models that are trained using textual data, such as emails, chats, or medical records, which frequently contain sensitive data or personally identifiable information. Thus, it is important for companies working in this field to assess the risk of customer data leakage and potential privacy breaches. This assessment is also required to proactively comply with data protection laws, such as the General Data Protection Regulation (GDPR), which enforces the need for privacy and demands protection against data breaches.

This thesis analyzes some of the major privacy threats that have emerged in recent research work from using ML models in the NLP domain. Particular attention is placed on text representation models, which convert texts into numerical vectors. The objective is to assess whether sensitive information can be inferred simply by accessing vector representations ("embeddings") of texts. For this purpose, we first review different text representation approaches, ranging from classical models to more recent ones based on deep learning. We then implement a recently proposed inversion attack and test it against the representation produced by the various models to analyze what type of information can be leaked and under which conditions recovery is possible.

Empirical results show that vectors that encode texts can reveal an astonishing amount of sensitive information, potentially compromising user privacy. For example, proper names can be recovered from vectors produced even by the most recent state-of-the-art deep learning models.

Sommario

Le tecnologie collegate al natural language processing (NLP) offrono potenti strumenti per l'analisi testuale, ad esempio suggeritori automatici per aiutare gli utenti ad essere più efficienti o classificatori che categorizzano i documenti in base al loro contenuto. Questi strumenti sono solitamente alimentati da modelli di machine learning (ML) addestrati su dati testuali come e-mail, chat o cartelle cliniche che spesso possono contenere dati sensibili o informazioni di identificazione personale. Per le aziende che operano in questo settore è quindi importante valutare il rischio di esposizione dei dati dei clienti e di conseguenti potenziali violazioni della privacy. Questa valutazione è necessaria anche per conformarsi in modo proattivo alle leggi sulla protezione dei dati, come il regolamento generale sulla protezione dei dati (GDPR), che impone il rispetto della privacy e la protezione contro le violazioni dei dati.

Questa tesi analizza alcune delle principali minacce alla privacy, emerse in recenti lavori di ricerca, derivanti dall'uso di modelli ML in ambito NLP. Particolare attenzione viene posta sui modelli di rappresentazione del testo, che vengono utilizzati per convertire i testi in vettori numerici. L'obiettivo è valutare se le informazioni sensibili possono essere dedotte semplicemente accedendo alle rappresentazioni vettoriali dei testi ("embedding"). A questo scopo, passiamo prima in rassegna diversi modelli di rappresentazione del testo, dai quelli classici a quelli più recenti basati sul deep learning. Successivamente, implementiamo un attacco di inversione recentemente proposto e applicato contro le rappresentazioni prodotte dai vari modelli, al fine di analizzare quale tipo di informazione può essere trapelata e in quali condizioni è possibile il recupero.

I risultati empirici mostrano che i vettori che codificano i testi possono rivelare una quantità sorprendente di informazioni, compromettendo potenzialmente la privacy degli utenti. Ad esempio, nomi propri possono essere recuperati da vettori prodotti anche dai più recenti modelli all'avanguardia basati sul deep learning.

Acknowledgements

I would like to express my gratitude to my supervisor Prof. Alberto Testolin for providing guidance and feedback throughout my thesis work.

I would like to extend my sincere thanks to Daniele Turato and Gioele Perin for their knowledge and expertise and who supported me during my internship. I also thank Siav and its staff for the internship opportunity.

Lastly, I would like to acknowledge my family, my parents in particular, and my friends for all their encouragement and support that motivated me during my master's degree studies.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

In the digital world in which we live, data have been defined as «the new oil» of the 21st century [1]: data, such as oil, are an unrefined resource that, when processed, can be transformed into a valuable asset. For companies working in the digital economy, data can be turned into a profitable resource, for example, by producing products tailored to their customers' needs. Subsequently, this can lead to satisfied customers and increased retention. For users, the insights generated from their data can be used to improve their productivity or the efficiency of their digital processes. Under the previous analogy, processed data can be viewed as combustible powering machine learning (ML) and deep learning (DL) models, which are capable of learning general underlying patterns from data to solve a plethora of different tasks spanning various domains.

In particular, this thesis focuses on ML and DL models applied to solving natural language processing tasks. Natural language processing (NLP) [2] is a subfield of linguistics and artificial intelligence (AI) that aims to provide machines with the ability to understand and derive meaning from text. In particular, NLP combines hand-crafted algorithms and statistical models. The statistical models, which can be implemented through machine learning or deep learning, allow machines to capture the hidden structure of language and understand its semantics.

NLP breaks down language understanding into several different tasks. Some of these tasks include text classification to categorize text according to some attribute (e.g., topic, sentiment); text similarity to determine how similar texts are; part of speech tagging to label each word in a text according to its role in the speech; text generation to generate fluent natural language, and many more. Many applications employ some of these tasks: document classification, machine translation, virtual

assistants, and search engines.

However, the data used to train these ML models often contain sensitive or private information. For example, text data used to train ML models in NLP can be represented by financial documents, email or chat messages, and medical records. These types of data naturally include sensitive attributes, such as personally identifiable information (PII), which could be used to identify and potentially cause harm to a specific individual.

Privacy is the right of individuals to control to what extent information about them is disclosed to others [3]. Many countries recognize privacy as a fundamental right, so they have established appropriate protection laws to safeguard it. For example, in the European Union (EU), the General Data Protection Regulation (GDPR) defines the EU's rules for collecting and handling personal data. In particular, the GDPR establishes the conditions and cases under which data about individuals can be collected and processed; it requires data protection policies, transparency with data owners, and technical and organizational measures to mitigate security risks. Any individual or organization who handles personal data in the EU must comply with the GDPR, and penalties are applied in the event of non-compliance with the legal terms of the GDPR.

The GDPR does not explicitly address AI, ML, and DL. However, many provisions in the GDPR are related and challenge how personal data are processed in these rapidly adopted frameworks [4]. For example, the GDPR requires performing an impact assessment to ensure that the rights and freedoms of data owners are not at risk when using *new technologies* to process personal data (Article 35(1), General Data Protection Regulation, 2018). Consequently, companies that use or want to adopt these frameworks must assess the risk of customer data leakage and potential privacy breaches derived from such frameworks.

1.2 Objective

This thesis work was carried out during an internship at Siav. Siav is an Italian software company specialized in document management software and digital processes. Siav operates in the enterprise content management (ECM) industry and some of the services it offers include electronic document management, electronic invoicing, and digital preservation. The objective of this thesis is two-fold and was defined together with Siav's research and development department.

First, we analyze the main threats to privacy documented in the literature that arise from using ML models to solve NLP tasks. These include: (1) the possibility of determining whether a particular individual with their data was part of the training

set of an ML model [5], [6]; (2) the extraction of text sequence segments appearing in the training data and that could potentially contain personally identifiable information [7], [8]; (3) the extraction of information from vectors encoding texts [9], [10].

The second objective, directly related to the previous one, is to empirically assess to what degree information can be extracted from vectors encoding texts. Therefore, we consider different text representation models that map raw texts into vectors of features. In particular, the analysis includes classical methods such as bag-of-words [11], latent semantic analysis [12], and Doc2Vec [13], up to more recent DL-based models such as Sentence-BERT [14]. We compare these models in a document classification task that represents a possible service provided by the company. To assess what type of information can be inferred by accessing the vector representations of texts, we implement an inversion attack against the representations produced by such models.

The scenario of interest that motivated the study is the following. A customer turns to a company that provides ML-based NLP services, such as document classification. The ML models are trained on the private data of the customer. To achieve privacy, the customer does not want to send their raw texts; rather, they prefer to produce vectorized representations of such texts and send them to the company. For this purpose, the company provides its customers with a text vectorization model, which should allow for building an accurate downstream model while the customer maintains their privacy. Figure 1.1 depicts the scenario considered. In particular, the customer locally produces vectorized representations of the raw texts, which are then sent to the company that produces the tailor-made model for the customer. An inversion attack is applied to the produced vectors to assess the risk of information leakage from vectorized representations. The inversion model is implemented as an ML model that infers the words appearing in the plaintext by simply accessing its vectorized representation. The output of the inversion model is a "bag of words" meaning that the order in which the words appear is not considered. It should be noted that even a combination of common words linked to a particular individual may constitute a privacy breach.

Empirical results show that classical models leak most of the information, while more advanced ones leak a reduced, but still considerable, amount of information, potentially compromising user privacy. For example, we show that more than 40% of the words[1] appearing in the raw text of a representation produced by Sentence-

[1]Stop words (i.e., words appearing with high frequency and expected to carry little information) and single character words were not considered.

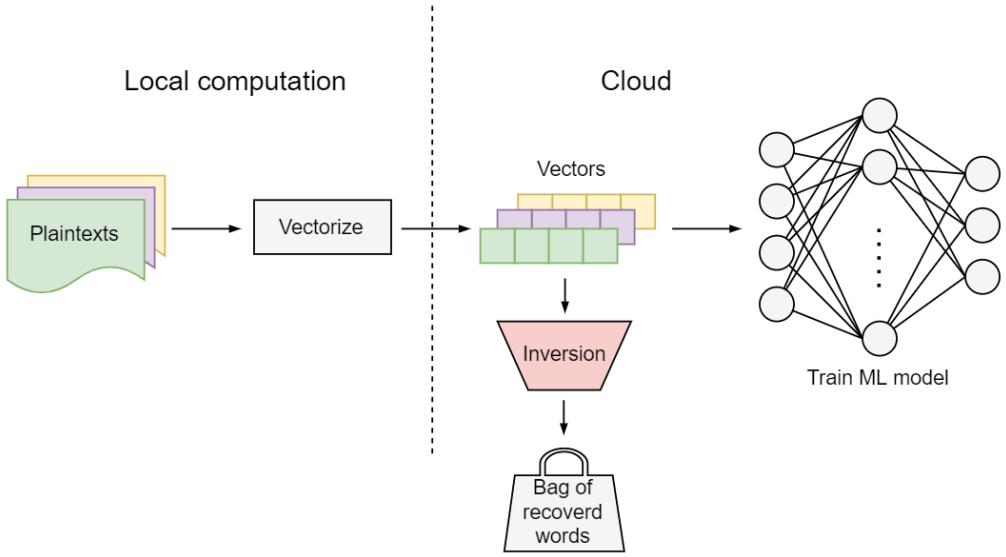


Figure 1.1: The scenario of interest. The customer locally produces vectorized representations of their private texts, which are then sent to the company that trains the downstream model. An inversion attack that aims to recover words by accessing the vectors is carried out against vectors produced by various representation models.

BERT can be reliably recovered. In addition, we show that full names can be retrieved from the vectors produced by Sentence-BERT. The results suggest that, despite some vectorizations limiting the amount of information that can be recovered, the analyzed strategy does not protect user privacy and appropriate mitigations are required.

1.3 Content Organization

The content of this thesis is organized as follows.

- Chapter 2 provides an overview of the encoding models chosen for the inversion attack. In particular, the models described are bag-of-words with and without feature hashing, latent semantic analysis, Doc2Vec, and Sentence-BERT.
- Chapter 3 describes some of the privacy-related risks found in the literature that arise from using ML models to solve NLP tasks. These include the fact that neural networks tend to memorize rare and unique details of the training data. Consequences of neural networks memorizing their training data include the extraction of training sequences from generative models, and the inference of whether a particular individual with their data was part of the model

training dataset using the so-called membership-inference attacks. Moreover, additional techniques are described to extract information from vectors that encode texts. Finally, the chapter also provides a high-level overview of differential privacy, a mathematically rigorous and sound approach for guaranteeing privacy in data analysis.

- Chapter 4 outlines the experimental methodology followed to validate the text representation models considered and to analyze the information leaked by the vectors such models produce. In particular, the models are validated using a classification benchmark that compares the considered models in a downstream classification task. The inversion attack, which, by accessing only the vectorized text representation, aims to recover the words appearing in the plaintext, is described in detail.

- Chapter 5 reports the experimental results obtained following the methodology outlined in Chapter 4. These include the results of the classification benchmark and the inversion results on the vectors encoding texts. In particular, the inversion results report the performance when inverting the vectorization of sentences and paragraphs, the accuracy in recovering names from vectors, and the scalability of the proposed approach related to the amount of data available.

- Chapter 6 reports the concluding remarks.

Chapter 2

Text Representation Models

Text analysis requires methods to encode any sequence of words, ranging from short sentences and paragraphs all the way up to long and structured documents, into a fixed-size vector of numerical features, which can then be fed to some downstream model to perform the analysis (e.g., text classification, document similarity, search query).

Over the years, many methods, with an increased degree of complexity that enabled state-of-the-art performance on such downstream tasks, were proposed to perform the encoding from raw text to numerical features. Nevertheless, sometimes simpler models, such as the bag-of-words, can stand their ground against more complex and fancy models depending on the task at hand and other external limitations (e.g., low memory footprint, no access to GPU accelerated computing); therefore, it is essential to know the trade-offs each model offers and choose accordingly for the given application.

This chapter provides an overview of the encoding models chosen for the analysis using a downstream classification task and the inversion attack described in Chapter 4. In particular, in Section 2.1 the simple yet effective bag-of-words model with its variants is reviewed; in Section 2.2, Doc2Vec, a shallow neural network model that paved the way to more advanced encoder-decoder models, is described; finally, in Section 2.3, the Sentence-BERT model, based on the deep architecture BERT, is described.

This chapter is far from including all the possible ways in which a sequence of words can be encoded into a fixed-length vector, as this is beyond the scope of this work. For a more comprehensive survey of the topic, the reader is invited to explore [15].

2.1 Bag of Words Models

The classical and most straightforward approach to represent a document is by means of a bag of words [11]: Given a vocabulary $\mathcal{V}$, a document d is represented with a frequency vector of dimension $|\mathcal{V}|$, where each word $w \in \mathcal{V}$ is associated with a unique index $i \in [0, |\mathcal{V}| - 1]$ in the vector. In this representation, documents are described only by the words they contain, and the information carried by the relative positioning of words is completely disregarded.

There exist different variations of this simple idea. For example, instead of considering a vector of word counts, binary entries can be used to encode only the presence or absence of words while ignoring their frequencies. Another variation consists in counting short n-length word sequences instead of only words, which yields the so-called bag of n-grams, which tries to capture some ordering information lost with the simple bag-of-words (for $n = 1$, the bag of n-grams is bag-of-words). However, such a variation has the disadvantage of an increased dimensionality over the simpler model, which explodes as n becomes larger (in practice, $n = 2, 3$).

$|\mathcal{V}|$ usually ranges from tens to hundreds of thousands, while documents typically have tens or hundreds of distinct words, and therefore the bag-of-words representation is inherently sparse. A possible way to compress such a representation is through latent semantic analysis (LSA), which is discussed in Section 2.1.2. In short, LSA is a technique that analyzes the relationship between documents and the words contained in them to find the underlying relationship between terms and concepts.

Assuming that each $w \in \mathcal{V}$ is encoded with a one-hot encoding, the vector resulting from the bag-of-words representation of a document d can be viewed as the sum of the one-hot vectors of each word appearing in d. This idea can be extended from one-hot vectors to word embeddings (e.g., those produced by Word2Vec, which is described in Section 2.2.1) — dense representation of words in which semantically related words are close in the vector space — to produce a document embedding as the centroid (usually preferred over summing) of its words' embeddings.

It is worth spending a few words regarding the choice of vocabulary since some of the same considerations can be applied to other representation models (e.g., Doc2Vec). Given a dataset of documents, $\mathcal{V}$ can be built considering all the distinct words appearing in the corpus's training split. If the model is used to represent other documents that contain words that did not appear in the training split, such words will be ignored in the representation. This is not a problem if the training split is large and varied and contains a rich set of words. However, in the training split, many uninteresting words may appear: for example, the so-called stop words

(i.e., words that appear with high frequency and therefore do not carry much information) or words that are orthogonal to the downstream task the representation is used for, such as diseases names or proper names in an e-mail classification setting. In the latter case, if this type of information is leaked by the representation model and an attacker can infer with a non-negligible degree of confidence that John Doe suffers from a given disease, this may represent a severe privacy threat. Therefore, the first line of defense against this kind of information leak could be not including such words in the vocabulary. Although this strategy seems simple, it is flawed because it cannot prevent all forms of privacy leakage.

In building the vocabulary, some parameters can be tuned: for example, all words whose document frequency exceeds a given threshold are deemed corpus-specific stop words and not included in the vocabulary, or on the other side of the spectrum, if a word appears only in less than a given number of documents, it can be ignored. Additionally, a hard threshold on the number of words in the vocabulary can be set to limit the representation dimensionality and consequently the complexity (measured in terms of parameters) of the downstream model that uses the representation.

The discrete bag-of-words model is usually complemented with the technique known as the *term frequency-inverse document frequency* (tf-idf) [11]. This method consists in reweighting the term frequencies $\mathrm{tf}(t, d)$ of the bag-of-words representation (i.e., the number of times each word, or n-gram of words, appears in the document d) by the inverse document frequency

$$\mathrm{idf}(t) = \log \frac{\mathrm{N}}{\mathrm{df}(t)} \tag{2.1}$$

where N is the total number of documents in the training split of the corpus and $\mathrm{df}(t)$ is the number of such documents that contain the term t. This reweighting aims to reduce the importance of words appearing very often and hence expected to carry little meaningful information about the document's actual content.

2.1.1 Hashing Trick

When dealing with large datasets, the simple bag-of-words model described above may become inefficient since it must store in-memory mappings between the tokens (e.g., words, n-grams of words or characters, etc.) and the corresponding integer indices, and the larger the corpus, the larger the vocabulary will grow.

A possible solution is the so-called *hashing trick* [11]. Instead of keeping the mappings between the tokens and the indices, the index of a given token t is determined from the hash of the token itself using $i = \mathrm{H}(t) \mod \mathrm{N}_f$, where $\mathrm{H}(\cdot)$ is the hash function of choice and N_f is the number of features that determines the

dimensionality of the representation.

Since the hash function may map different, unrelated tokens to the same vector position, it is important to choose N_f appropriately: it should neither be too small so that many collisions occur nor too large, which, in turn, would make the document representation unnecessarily large. Additionally, it is possible to choose a signed hash function, and the sign of the token hash determines the sign of the value stored at the computed index so that collisions are likely to cancel out rather than accumulating errors.

Other than the low memory footprint and the increased speed, a side effect of this stateless[1] technique is that no vocabulary is built, and the mappings from token to index cannot be easily inverted.

2.1.2 Latent Semantic Analysis

With the bag-of-words model, a collection of documents $\mathcal{D}$ can be represented by a $|\mathcal{D}| \times |\mathcal{V}|$ document-term matrix C where each row represents a document and each column represents a term. As previously observed, this matrix can be very large, with tens or hundreds of thousands of rows and columns.

The *singular value decomposition* (SVD) from the linear algebra toolbox comes in handy when trying to obtain a low-rank approximation of such large matrices. The low-rank approximation of the document-term matrix obtained using SVD is known as *latent semantic analysis*[2] (LSA) [12], [16] because it transforms the document-term matrix into a "semantic space" of lower dimensionality.

Mathematically speaking, to obtain a low-rank approximation C_k of rank at most k of the document-term matrix C with rank $r \leq \min\{|\mathcal{D}|, |\mathcal{V}|\}$ minimizing the *Frobenius norm* of the error $\epsilon = ||C_k - C||_F = \sqrt{\sum_{i=1}^{|\mathcal{D}|} \sum_{j=1}^{|\mathcal{V}|} (C_{k,ij} - C_{ij})^2}$, the following procedure should be followed:

1. Using the SVD, factor C as the product of three matrices: $C = U\Sigma V^\intercal$. Σ is a $r \times r$ diagonal matrix whose values, called singular values and ordered in non-increasing order, are the square of the eigenvalues of the real, symmetric matrix $CC^\intercal$ (or equivalently of $C^\intercal C$). U and V are rectangular matrices of dimension $|\mathcal{D}| \times r$ and $|\mathcal{V}| \times r$ respectively, whose columns are orthogonal and normalized to unit length[3].

[1] The hashing vectorization model is stateless in the sense that once $\mathrm{H}(\cdot)$ and N_f are defined, no other additional information is required to compute the representation.

[2] In the field of information retrieval LSA is also known as *latent semantic indexing* (LSI).

[3] What has been described is sometimes called reduced or truncated SVD. In normal SVD, Σ is a $|\mathcal{D}| \times |\mathcal{V}|$ matrix where all the entries outside the $r \times r$ sub-matrix's diagonal are zeros; U is a $|\mathcal{D}| \times |\mathcal{D}|$ matrix whose columns, called left-singular vectors, are the eigenvectors of $CC^\intercal$; V is a

2. Obtain Σ_k by replacing by zeros the $r - k$ smallest singular values on Σ's diagonal and U_k, V_k by omitting the rightmost $r - k$ columns that would be multiplied by zeros.

3. Compute the rank-k approximation of C as $C \approx C_k = U_k \Sigma_k V_k^\intercal$.

This procedure yields the matrix of rank k with the lowest possible Frobenius error ϵ, which is equal to the largest singular value set to zero: thus, the larger k, the smaller the error of the approximation.

The low-rank approximation of C yields a new representation $C' = U_k \Sigma_k$ for each document in a k-dimensional space defined by the k principal eigenvectors (corresponding to the largest eigenvalues) of $CC^\intercal$. $CC^\intercal$ is a $|\mathcal{V}| \times |\mathcal{V}|$ matrix whose (i, j) entries measure the co-occurrence, in the collection of documents associated with the document-term matrix C, between the terms i and j.

To map a new collection of documents with the document-term matrix $\bar{C}$ in this new representation, it suffices to compute $\bar{C}' = \bar{C} V_k$.

In addition to compressing the representation provided by the bag-of-words, LSA *should* deal better with the phenomena of *synonymy* (different words with the same meaning, e.g., "car" and "automobile") and *polysemy* (words with multiple meanings, e.g., "bank" of the river and "bank" the institution).

2.2 Neural Network Based Embeddings

In the bag-of-words representation, the semantics of the words are not taken into account since words are treated as indices in a vocabulary. For example, using the one-hot encoding, the words "guitar", "violin", and "pizza" are all equally distant.

Algorithms based on neural networks and trained on large amounts of unstructured text were proposed to overcome this limitation and learn lower dimensional word representations (compared to one-hot encoding), which also encode the semantics of the words.

Word2Vec [17] is a prime example of these neural network based word representation models that embed words in a low-dimensional continuous vector space in which words with similar meanings (based on context, i.e., nearby words) are close to each other while words with different meanings are far apart. Such representations allow one to perform linear operations like $v(\text{"king"}) - v(\text{"man"}) + v(\text{"woman"}) \approx v(\text{"queen"})$ where $v(\cdot)$ is the neural embedding produced by Word2Vec.

$|\mathcal{V}| \times |\mathcal{V}|$ matrix whose columns, called right-singular vectors, are the eigenvectors of $C^\intercal C$. From the structure of Σ, it is clear that there is an equivalence between the two decompositions.

Paragraph vector [13], also known as Doc2Vec, was subsequently proposed as an extension of Word2Vec to learn the embeddings of words sequences irrespective of the granularity of the sequences themselves, which can equally be n-gram of words, sentences, paragraphs, or documents.

Since many of the building blocks Doc2Vec builds upon are based on Word2Vec, Section 2.2.1 provides a brief overview of Word2Vec, while Section 2.2.2 describes the Doc2Vec model.

2.2.1 Word2Vec

Word2Vec [17] is a neural network based approach to learning high-quality word embeddings that capture the semantics of words.

The Word2Vec word embeddings are learned by training a 1-hidden-layer neural network on the synthetic task of predicting a context word w_O given the input word w_I. In the neural network, the one-hot encoding of w_I is projected into a hidden representation v_{w_I}, which is then used in the output layer to predict the probability distribution of context words according to the softmax operation:

$$P(w_O|w_I) = \frac{\exp(v'^{\intercal}_{w_O} v_{w_I})}{\sum_{w \in \mathcal{V}} \exp(v'^{\intercal}_{w} v_{w_I})} \tag{2.2}$$

where v_w and v'_w are the hidden and output vector representations of the word w in the vocabulary $\mathcal{V}$[4].

After training, the hidden representation v_w, into which each word is projected, is used as the word's embedding.

The learning objective is formulated as the maximization of $\log P(w_O|w_I)$. However, such a formulation is impractical, as this would make the computation of $\nabla \log P(w_O|w_I)$ $\Omega(|\mathcal{V}|)$ due to the denominator in Equation 2.2.

The authors introduced the negative sampling technique [18] which approximates $\log P(w_O|w_I)$, making the computation feasible even for large vocabularies since only a subset of weights is updated at each back-propagation step.

More in detail, with negative sampling, the learning objective is to maximize the similarity between w_I and w_O, measured in terms of the dot product, while

[4]That is to say v_w is the i-th column in the first weight matrix (one-hot to hidden neurons projection) and v'_w is the i-th row in the second weight matrix (mapping from hidden neurons to distribution in $\mathbb{R}^{|\mathcal{V}|}$) where i is the index of the word w.

minimizing the dot product of w_I and randomly sampled *negative* words:

$$\log P(w_O|w_I) \approx \log \sigma(v'^{\intercal}_{w_O} v_{w_I}) + \sum_{i=1}^{k} E_{w_i \sim P_n(w)} \left[\log \sigma(v'^{\intercal}_{w_i} v_{w_I})\right] \tag{2.3}$$

where $\sigma(\cdot)$ is the sigmoid function, k is the number of negative samples, and $P_n(w)$ is the noise distribution.

The authors found that the unigram distribution raised to the 3/4rd power, i.e., $\frac{c(w_i)^{3/4}}{\sum_{w \in \mathcal{V}} c(w)^{3/4}}$ where $c(w)$ is the count of the word w, in which the most frequent words are more likely to be sampled as negative words, is the most effective choice for the noise distribution.

From Equation 2.3, it can be seen that with negative sampling, the task becomes to distinguish the target word w_O from k randomly sampled noise words using logistic regression.

An additional improvement to speed up training and improve the quality of the learned vectors for rare words [18] consists in subsampling frequent words (e.g., "the", "a", "in") that are usually less informative than rarer words. Each word w_i is discarded from the training set according to the probability:

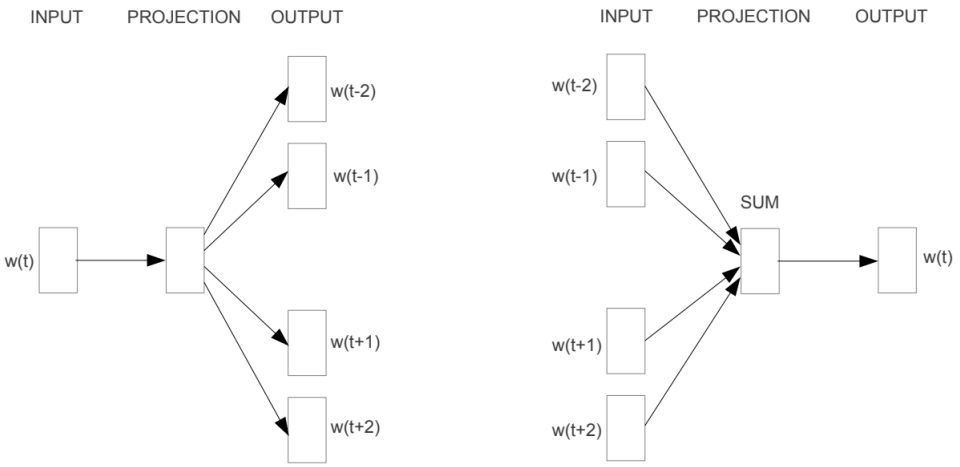
$$P(w_i) = 1 - \sqrt{\frac{t}{c(w_i)}}$$

where $c(w_i)$ is the count of word w_i and t is a hyperparameter: words whose frequency exceeds t are aggressively subsampled.

There exist two variants of Word2Vec: *skip-gram* and *continuous bag-of-words* (cbow) which differ in the role assumed by w_I and w_O.

In skip-gram, w_I is the central word in a window of words that slides along the training text, while w_O is a context word surrounding w_I and within the window. In cbow, w_I consists of multiple context words within the sliding window that are summed (or averaged) together via vector addition and are used to predict the central word of the window w_O. The number of context words on the left and right of the central word is the window size hyperparameter. Figure 2.1 illustrates the difference between the two models.

Word2Vec is built on the *distributional hypothesis* of linguistics, according to which words that are used and occur in the same contexts tend to bear similar meanings. This can be seen from the fact that the training objective in Word2Vec is to learn word representations that are useful for predicting the surrounding words in a sentence. Therefore, if two different words (e.g., "car" and "automobile") tend to appear in similar contexts (i.e., with similar surrounding words), the representation



(a) Skip-gram Word2Vec. (b) Continuous bag-of-words Word2Vec.

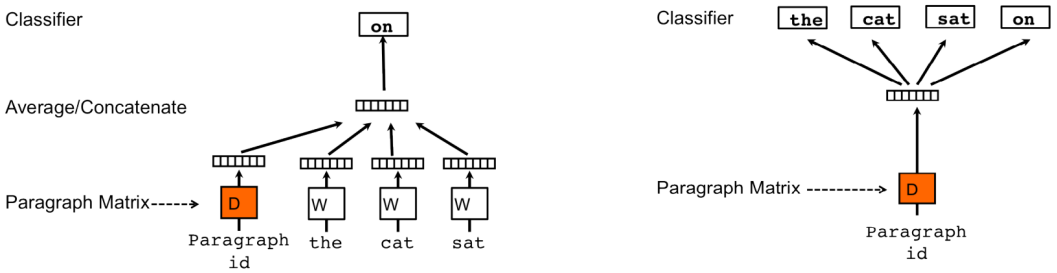
Figure 2.1: Word2Vec model architectures [17]. Word2Vec embeddings are learned by training a neural network to predict a context word given an input word.

learned by Word2Vec for the two words will be close in the vector space since they both must produce a similar distribution of context words.

2.2.2 Doc2Vec

Paragraph Vector [13], later popularized with the name Doc2Vec, was introduced as a natural extension of Word2Vec to encode variable-length pieces of text, such as sentences and documents [19], into fixed-length feature representations.

Similarly to Word2Vec, it is an unsupervised framework that learns continuous vector representations by being trained on the synthetic task of predicting words in a text and comes in two variants: *distributed memory* (dm) and *distributed bag-of-words* (dbow) which are depicted in Figure 2.2.



(a) Distributed memory. (b) Distributed bag-of-words.

Figure 2.2: Doc2Vec model architectures [13]. Doc2Vec embeddings are learned by training a network to predict words given the vector of a text, called the paragraph vector.

In Doc2Vec, each variable-length text is mapped to a unique vector called the *paragraph vector*, represented by a column in the matrix D, which is used to perform the prediction task; similarly, each word is represented by a column in the matrix W.

In dm-Doc2Vec, the paragraph vector is concatenated with a fixed-length context of words sampled from a sliding window over the text to predict a single word following the context. The paragraph vector is shared among all contexts generated from the same text, while the matrix of word embeddings W is shared among all texts. Therefore, the paragraph vector acts as a memory that encodes what is missing from the current context (or the topic of the text). The architecture of dm-Doc2Vec (Figure 2.2a) is reminiscent of that of cbow-Word2Vec (Figure 2.1b) with the difference in the added information carried by the paragraph vector and the order of the words in the context, which is taken into account since the words are concatenated[5].

Dbow-Doc2Vec (Figure 2.2b) is the simpler model, which, despite the name, resembles skip-gram Word2Vec (Figure 2.1a): the context words are ignored and, given only the paragraph vector, the model is trained to predict words randomly sampled from the text.

Both the improvements proposed for Word2Vec, negative sampling, and frequent word subsampling described in Section 2.2.1 are also applied in Doc2Vec.

Paragraph and word vectors are trained using stochastic gradient descent (SGD). After the training is completed, the resulting paragraph vectors in D can be used as embeddings of the training documents.

To infer the embedding of a new, unseen document, an inference stage is required: the paragraph vector for the new document is randomly initialized, and while keeping all the parameters of the network fixed (e.g., word vectors W and logistic regression parameters), the weights of the new paragraph vector are optimized to map the new embedding in the same space the training documents' embeddings lie.

It is worth noting that the number of epochs and the learning rate during training and the inference stage need not be the same. Sometimes, a larger number of epochs or a smaller learning rate in the inference stage may yield better representations. It is also important to note that, given the inherent randomness of the inference stage, if the same document is encoded multiple times, the embeddings returned will not be exactly the same each time but will still be close in the embedding space.

Although the authors claim that dm-Doc2Vec is the better model, producing

[5]In the original paper [13] the authors experiment with concatenation; however, averaging, which loses the ordering information, is also possible.

text representations that achieved higher performance when used in a sentiment analysis classification setting, others were unable to replicate the same results and found that the simpler dbow-Doc2Vec produces better representations [20].

The recipe for improving the dbow model, introduced by the Gensim library [21], is to, instead of randomly initializing the word vectors (logistic regression parameters), simultaneously learn them in a skip-gram fashion[6]: for each text example, both the paragraph vector of the whole text and the word vectors over each sliding context window are trained. A possible motivation for this is that it places both the dbow paragraph and word vectors into the same space.

2.3 Transformer Based Language Models

In recent years, a new paradigm has emerged based on building large language models with millions of learnable parameters that are trained with unsupervised objectives on massive datasets. Once these large language models are trained, they are able to capture complex language patterns and can be used to generate fluent, natural language or transfer their knowledge to various downstream tasks to achieve state-of-the-art performance. Some examples of such language models include OpenAI's GPT-3 [22] and Google's BERT [23].

In particular, pre-trained BERT can be used as a text features extractor to encode sentences (or, more generally, pieces of text) into dense vector representations. However, in [14], it was found that the model does not produce quality embeddings without proper fine-tuning. Therefore, sentence-BERT [14] was introduced to overcome this limitation and adapt BERT to produce semantically meaningful embeddings that, for instance, can be used to perform text similarity search queries.

After a brief detour in Section 2.3.1 reviewing language modeling, Section 2.3.2 overviews the BERT model, while Section 2.3.3 describes Sentence-BERT.

2.3.1 Language Modeling

Language models (LMs) are an important building block in many NLP tasks. Different unsupervised objectives exist to train these models depending on the exact architecture of the LM. A common approach is *next-step prediction* [8] which builds a generative model based on the underlying distribution of tokens (e.g., words) in

[6] Alternatively, pre-trained Word2Vec word embeddings can be used [20].

the training corpus

$$P(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} P(x_i | x_1, \ldots, x_{i-1}) \tag{2.4}$$

where x_i for $i = 1, \ldots, n$ is a sequence of tokens from the vocabulary $\mathcal{V}$.

LM based on neural networks estimate this distribution with $f_\Theta(x_i | x_1, \ldots, x_{i-1})$ which indicates the likelihood of the token x_i when the neural network f with parameters Θ is provided the prefix $x_1, \ldots, x_{i-1}$. In particular, in this setup, the training objective is to maximize the log-probability in Equation 2.4 of the training dataset. This translates into minimizing the following loss function:

$$\mathcal{L}(\Theta) = -\log \prod_{i=1}^{n} f_\Theta(x_i | x_1, \ldots, x_{i-1}) \tag{2.5}$$

over each sample (e.g., sentences, documents) in the training dataset.

These language models, trained with the next-step prediction objective, are usually called autoregressive LMs. A trained autoregressive LM can be used to generate new text: by feeding some context $x_1, \ldots, x_i$ (potentially empty) and iteratively sampling $\bar{x}_{i+i} \sim f_\Theta(x_{i+i} | x_1, \ldots, x_i)$ and feeding back $\bar{x}_{i+i}$ into the model to sample $\bar{x}_{i+2} \sim f_\Theta(x_{i+2} | x_1, \ldots, x_i, \bar{x}_{i+i})$ until a stopping criterion is reached.

While recurrent neural networks (RNNs) used to be a popular choice for building neural-based LMs, the most recent LMs are based on the Transformer [24]: a deep architecture with millions of parameters that requires a huge corpus to be trained effectively.

An example of a state-of-the-art autoregressive LM is the OpenAI GPT-3 model, which is trained on data scraped from the Web and can generate fluent natural language. On the other hand, Google BERT is a different language model known as masked LM, as will be detailed in Section 2.3.2.

2.3.2 BERT

BERT (Bidirectional Encoder Representations from Transformers) [23] is a deep learning model that falls in the paradigm of building a powerful LM by pre-training on a huge corpus and then transferring the acquired knowledge to various downstream tasks (e.g., text classification, question answering, sentence tagging, sentence-pair regression) by fine-tuning the entire model with minimal architectural modifications. This paradigm allowed for many breakthroughs in the NLP field, enabling to reach state-of-the-art performances in an array of natural language understanding tasks: when BERT was first published, it improved previous results on eleven NLP

tasks.

BERT is based on the Transformer encoder architecture [24], a deep learning model which allows the processing of sequential data, such as texts, in parallel, without a predefined order (e.g., left to right) as it happens in RNNs.

The Transformer encoder (Figure 2.3) consists of two sub-layers: a multi-headed self-attention layer followed by a feed-forward neural network.

The input of the encoder is a sequence of dense vector representations. For each element in the sequence, the self-attention layer produces a new intermediate representation by accessing all input sequence elements but only *attending* on those that matter to encode the current element. The weights that govern what elements the encoder should attend to while processing a given sequence element are learned during the training process.

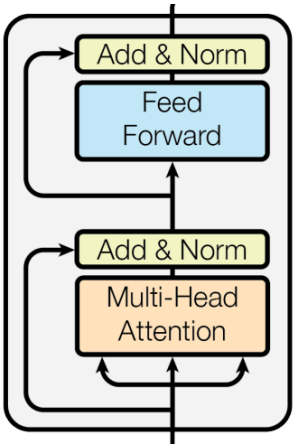


Figure 2.3: The Transformer encoder architecture [24].

The self-attention mechanism is multi-headed because, for each item in the sequence, it produces A different representations (corresponding to the attention heads) focusing at different positions in the input sequence. The representations of the different heads are then combined in a "summary" representation.

Finally, the representations are passed as input to the feed-forward neural network that applies the same computations to all sequence elements, producing a sequence of hidden representations, each with H units corresponding to the encoder output.

BERT uses a stack of L Transformer encoders (Figure 2.4).

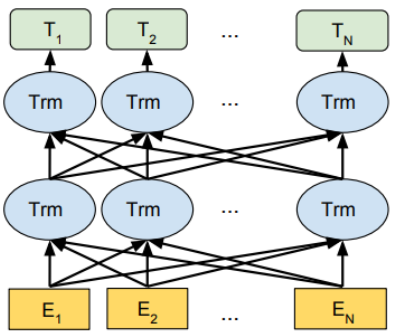


Figure 2.4: The BERT model architecture [23].

The input sequence, which may equally be a single sentence or a pair of sentences separated by the `[SEP]` token, is preappended with the special classification token `[CLS]` and tokenized into smaller units (words or sub-words) according to a prepared vocabulary.

Each token is mapped to its embedding, which is augmented by summing it with the segment and positional embedding. The segment embedding indicates to which sentence each token belongs, while position embed-

ding, which follows a specific pattern, encodes the position of each token in the sentence. The resulting embedding E_i for each token i in the input sequence is then propagated through the Transformer encoder stack, producing the hidden vector $T_i \in \mathbb{R}^H$.

BERT uses WordPiece embeddings with a vocabulary of 30k tokens so that unknown words are split into sub-words. Additionally, every character appears as a sub-word in the vocabulary: this allows BERT to tokenize any unknown word into multiple known sub-words, including single characters if needed, avoiding the problem of out-of-vocabulary words (OOV)[7].

BERT comes in two variants: $\text{BERT}_{\text{BASE}}$ ($L = 12$, $H = 768$, $A = 12$) and $\text{BERT}_{\text{LARGE}}$ ($L = 24$, $H = 1024$, $A = 16$) which have, respectively, 110M and 340M total parameters.

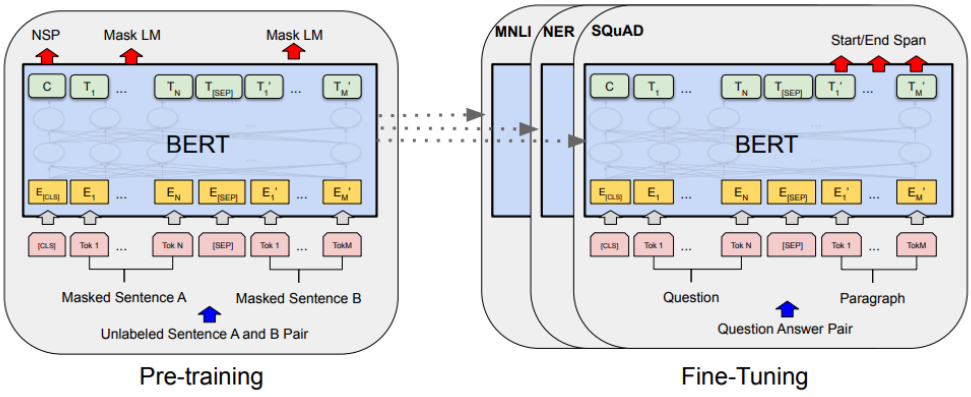


Figure 2.5: Pre-training and fine-tuning of BERT [23]. First, BERT is pre-trained on a large corpus with two sub-tasks: masked language modeling and next sentence prediction. Once BERT is pre-trained, the knowledge acquired can be extended on various downstream tasks by fully fine-tuning the model.

From the above description, it is clear that BERT is not a traditional language model since it produces representations conditioning on both left and right contexts across all layers. Due to this bi-directionality, a different pre-training strategy was required, given that bidirectional conditioning in a multi-layer setting would allow each word to see itself indirectly, making the prediction trivial. BERT is therefore pre-trained on two tasks (left part of Figure 2.5): masked language modeling (MLM) and next sentence prediction (NSP).

The objective of MLM is to randomly mask a fraction of the input tokens and then predict those masked tokens by feeding the corresponding hidden vectors into a softmax layer over the vocabulary (as in standard LM). More in detail, 15% of

[7]Even though BERT uses WordPiece embeddings, the vocabulary still contains the `[UNK]` token.

the token positions are selected at random. Each selected position has its token replaced with `[MASK]` 80% of the times, a random token 10% of the times, or is left unchanged. The hidden representation T_i produced for each selected position i is used to predict the original token with cross-entropy loss.

The objective of NSP is to predict whether two sentences have a logical, sequential connection or their combination is simply random. This pre-training objective helps the model understand the relationship between two sentences, which is not captured by language modeling and is especially beneficial in some downstream tasks (e.g., question answering). Each pre-training example consists of a pair of sentences `A` and `B` with `B` the actual next sentence 50% of the time and a random sentence the other half of the time. The prediction is performed using the hidden representation C of the `[CLS]` token.

Once the model is pre-trained on a large corpus[8], its knowledge can be transferred to many downstream tasks with minimal changes to the architecture and by fine-tuning the model end-to-end (right part of Figure 2.5). For example, in a classification scenario (e.g., classify whether an incoming e-mail is spam or not), the input consists of the text (e-mail corpus) as sentence `A` (and empty sentence `B`) while at the output, the hidden representation of the `[CLS]` token, or the mean pooling over the last layer's hidden representations, is fed into an output layer for classification.

Pre-trained BERT can also be used to extract word embeddings: the last hidden representation T_i (or the concatenation of the hidden representations of the highest layers) produced for each input token i can be used as a feature representation for a downstream task. The authors apply this strategy with a named entity recognition task (NER), achieving performances not too far from fine-tuning the entire model. The word embeddings thus obtained are called *contextualized* because the word embedding varies according to the meaning of the sentence in which the word occurs. For example, the embedding for the word "bank" will encode different information when produced from the two sentences: "I went to the bank to withdraw some money" and "While fishing, I sat on the bank of the river". Similarly, sentence embedding can be produced by performing a mean pooling operation on the hidden representations of the last hidden layer or by taking C as a summary representation of the entire sentence.

[8]The original BERT was pre-trained on the combination of the English Wikipedia and the BoookCorpus, an extensive collection of 11k books.

2.3.3 Sentence-BERT

Sentence-BERT [14] was proposed as an extension of BERT to derive semantically meaningful sentence embeddings so that semantically similar sentences are assigned to close representations in the vector space.

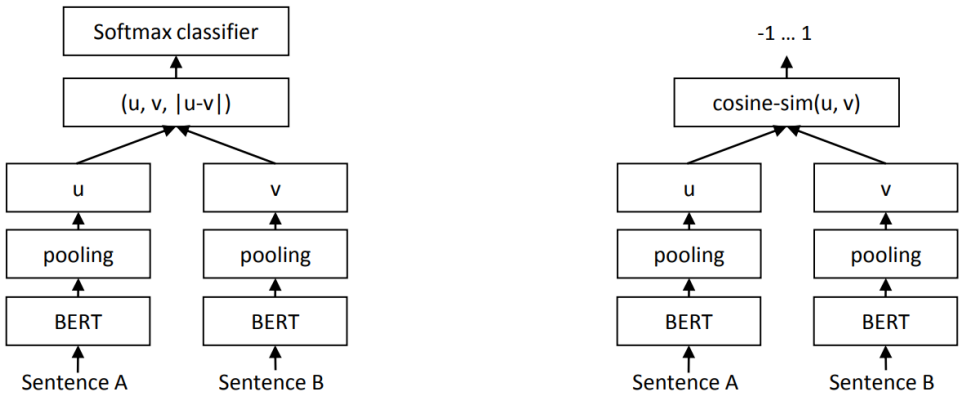
In particular, the authors of Sentence-BERT found that the representations produced by pre-trained BERT for the `[CLS]` token or by element-wise mean pooling over the last layer's representations do not lend themselves to be compared using similarity measures like the cosine similarity, defined as:

$$sim(u, v) = \frac{u^\intercal v}{\|u\|_2 \|v\|_2} \tag{2.6}$$

where u and v represent the embeddings of the sentences and $\|\cdot\|_2$ their ℓ_2 norm.

Another limitation of BERT is that to solve sentence-pair tasks, it uses a cross-encoder that takes as input both sentences. This may become inefficient for various sentence-pair tasks due to the many possible input combinations. For example, finding the pair of sentences with the highest similarity in a collection of $n = 10$k sentences would require $n(n-1)/2 \approx 50$M inference computations instead of only n if the produced embeddings could be effectively compared using a similarity measure.

To overcome these limitations, Sentence-BERT adapts the BERT architecture, using siamese and triplet network structures with tied weights, and fine-tunes it to produce quality embeddings. This unlocks the possibility of using rich BERT embeddings for tasks such as semantic similarly search and clustering.



(a) Classification objective

(b) Regression objective

Figure 2.6: Sentence-BERT siamese architectures [14]. Sentence-BERT uses siamese architectures to fine-tune BERT to produce quality embeddings. The exact architecture depends on the training data available.

The structure of the network and the objective function depending on the avail-

able training data:

- Classification objective function (Figure 2.6a): the sentence embeddings u and v are concatenated with their element-wise difference $|u - v|$ and the resulting vector is fed into a softmax classification layer. The weights are optimized according to the cross-entropy loss.
  This architecture can be used for fine-tuning with a natural language inference (NLI) dataset, which for each pair of sentences assigns a label that determines whether the second sentence is an *entailment*, *contradiction*, or *neutral* with respect to the first one.

- Regression objective function (Figure 2.6b): the cosine similarity between the two sentence embeddings u and v is computed, and the weights are optimized according to the mean squared error (MSE) loss.
  This architecture can be used for fine-tuning with a semantic textual similarity (STS) dataset, a collection of sentences annotated with a score that denotes how semantically similar the two sentences are.

- Triplet objective function: the network is optimized so that the distance between an anchor sentence a and a positive sentence p is smaller than the distance between a and a negative sentence n. Mathematically, the following function is minimized:

  $$\max\left(\|s_a - s_p\| - \|s_a - s_n\| + \epsilon, 0\right)$$

  where s_x is the embedding of the sentence x, $\|\cdot\|$ a distance metric and ϵ a margin.

In [14] the authors find that by fine-tuning BERT with the classification objective function on the combination of the Standford NLI (SNLI) and Multi-Genre NLI (MultiNLI) datasets, the Spearman rank correlation coefficient between the cosine similarity of the sentence embeddings and the gold labels for various STS datasets significantly improved over other sentence embedding techniques (InferSent and Universal Sentence Encoder). Instead, only taking the `[CLS]` token or the element-wise mean pooling without the described fine-tuning procedure achieved results lower than the simple averaging of word embeddings (GloVe embeddings).

Furthermore, they evaluated the embeddings produced by the fine-tuned model using SentEval [25], a toolkit used to assess the quality of text embeddings by feeding them into a logistic regression classifier trained on various tasks (e.g., sentiment analysis). In this scenario, even though the improvement is not as significant as that registered when the embeddings are used to compute similarities, they still achieved

a decent performance gain with respect to the [CLS] token or the element-wise mean pooling. A possible explanation for why pre-trained BERT embeddings yield such poor performance when used with similarity measures is that the similarity measures treat all dimensions equally. In contrast, when the embeddings are used with a logistic classifier, the dimensions are weighted appropriately. Furthermore, in both situations, they found that the mean pooling over the hidden representations consistently achieved better performance than taking the hidden representation of the [CLS] token.

Chapter 3

Major Privacy Risks in ML Models for NLP

Machine learning models are generally trained on users' private data, such as text messages, emails, and medical records. However, despite being trained on potentially sensitive information, such models can be released publicly, allowing anyone to interact with them. For example, Google's Smart Compose [26], a deployed RNN-based LM used for text completion, is trained on millions of users' email messages. Gmail users use Smart Compose to compose their emails; therefore, anyone with a Google account can interact with the model trained on private email messages.

However, making public models trained on sensitive data for anyone to query raises security and privacy concerns. Recent research work showed that this is indeed the case. For example, neural networks, on which most state-of-the-art models are based, were shown to memorize unique and rare secrets from their training data (see Section 3.1). This, in turn, opens the door to many privacy breaches, some of which are explored in this chapter.

Possible consequences of neural networks memorizing their training data include the extraction of training data sequences from generative models such as LMs (see Section 3.2) and the inference of whether a particular individual with their data was part of the model training dataset using the so-called membership inference attacks (see Section 3.3). Other possible attacks analyzed in the literature and on which the experimental part of this thesis focuses include ways to extract information from vectors that encode texts (see Section 3.4).

Techniques to mitigate these and other privacy risks are an active research frontier. A promising technique that is being studied and implemented is differential privacy (see Section 3.5), which is a mathematically rigorous and sound approach to guaranteeing data privacy.

As recent events have shown, memorization in neural networks with all its consequences is not just an academic problem. For example, in June 2021, GitHub released Copilot[1], a code completion tool powered by a GPT-type LM called OpenAI Codex [27] and trained on the source code publicly available on GitHub. By simply asking Copilot to write the code to solve a fast inverse square root, it reproduced verbatim the code lines from the Quake III implementation and the (incorrect) copyright license [28].

This chapter discusses prominent forms of privacy leakage that have emerged in recent years when using ML models to solve NLP tasks. However, it is not a comprehensive study but rather an introductory treatment that motivates this study and covers (from a high level) a possible route to mitigate such information leakage.

3.1 Memorization in Neural Networks

Machine learning models should learn to *generalize* the underlying patterns of their training data: models need not memorize any of their training data but should be able to extract knowledge from them. The acquired knowledge can then be used to perform tasks (e.g., text generation and text classification) on newly unseen data while still achieving performance close to that achieved on training data[2]. However, learning must involve some form of memorization [7]. For example, spelling in character-level LMs or what features (e.g., the words in a text) signal whether a sample belongs to a given class in a classification task.

The problem arises in the case of *unintended memorization*, when the model may expose out-of-distribution training data, such as the number of John Doe's credit card, which are unrelated to the learning task at hand and do not improve the overall accuracy of the model. If unintended memorization occurs, an attacker can exploit it to infer the membership of a data sample to the model training data, and this can represent a serious privacy breach in case the training data contain sensitive or private information.

To highlight the privacy implications of unintended memorization, consider the following motivating example. A text composition assistant uses a generative model for sentence completion. If the model was trained on users' data that happened to contain rare but sensitive information, the model should never reveal such information to the users interacting with the composition assistant. For example, if a user's

[1] `https://github.com/features/copilot/`

[2] When a model achieves performance similar between training and held out data is usually said not to overfit its training data as the generalization error is small.

text with the prefix "my credit card number is..." appeared in the training data, the model should never predict the exact credit card number of the user.

Unfortunately, Carlini et al. [7] showed that neural networks *do* memorize their training data. Toward this end, they proposed a testing methodology to quantify to what extent such unintentional memorization occurs in generative sequence models. Furthermore, they showed that, unless appropriate care is taken, with well-crafted attacks, in some cases it is possible to extract such secrets even if they have been seen only once or a few times during training. Section 3.1.1 describes the methodology proposed in [7] to quantify unintended memorization, while Section 3.1.2 summarizes some of [7] key findings.

3.1.1 Revealing Memorization in Generative Models

Carlini et al. [7] devised a pragmatic approach to quantitatively assess the risk that generative sequence models unintentionally memorize unique or rare training data sequences.

The procedure consists of the following three steps:

1. Injecting into the training dataset random sequences called canaries[3]. The canaries act as artificial *secrets* the model should not reveal.

2. Training the model using the augmented dataset.

3. Assessing to what degree the model memorized the canaries using a metric called *exposure*.

The canaries have a known fixed format that specifies how they are formed by sampling random values r from a space $\mathcal{R}$. For example, a possible canary format could be $s =$ "the random number is □□□□□□□" with $\mathcal{R}$ the space of digits from 0 to 9. Using the same notation as the authors, $s[r]$ denotes the format s with values filled in from randomness r and $s[\hat{r}]$ denotes the canary selected by sampling a random value $\hat{r}$ uniformly at random from $\mathcal{R}$. For example, a possible instantiated canary with the specified format could be $s[\hat{r}] =$ "the random number is 2022568"

To define the exposure metric, the definitions of log-perplexity [7], [8] and rank [7] are required.

[3]The name is reminiscent of "canary in a coal mine". Miners used to bring canaries with them because dangerous gases would kill the canaries and alert the miners that they must leave the tunnels.

Definition 3.1.1 (Log-perplexity)**.** Given a generative sequence model f_Θ with parameters Θ, the *log-perplexity* of the sequence $x = x_1, \ldots, x_n$ is

$$\begin{aligned}\mathcal{P}_\Theta(x) &= -\log_2 P(x_1, \ldots, x_n | f_\Theta) \\ &= \sum_{i=1}^{n} -\log_2 f_\Theta(x_i | x_1, \ldots, x_{i-1})\end{aligned}$$

The log-perplexity is inversely proportional to the likelihood of the data sequence $x_1, \ldots, x_n$ assigned by the generative sequence model f_Θ. Intuitively, perplexity measures how surprised the model is to see a given sequence. That is, if the perplexity is high, it means that the model is surprised by the sequence, whereas a low perplexity indicates that the sequence is likely a common, non-surprising sequence. For example, we would expect a trained LM to have a lower perplexity for the natural sequence "Mary had a little lamb" with respect to four random words "correct horse battery staple". This would generally be the case even if the latter nonsense sequence appeared in the training dataset while the former did not. This is because LMs are generally able to capture the true underlying distribution of natural language. Therefore, the log-perplexity measure alone to detect unintended memorization is ill-suited.

On the other hand, comparing the log-perplexity of the inserted canary $s[\hat{r}]$, on which the model was trained, and all other non-selected canaries with the same format not seen during training can be used to assess unintended memorization. In particular, the exposure metric uses this relative approach by using the rank of a canary.

The rank of a specific instantiated canary is defined as its index in the list of all possible canaries ordered according to their empirical perplexity assigned by the model. In mathematical terms, we can define the rank as follows.

Definition 3.1.2 (Rank)**.** Given a generative sequence model with parameters Θ, the *rank* of a canary $s[r]$ is

$$\text{rank}_\Theta(s[r]) = |\{\, r' \in \mathcal{R} \mid \mathcal{P}_\Theta(s[r]') \leq \mathcal{P}_\Theta(s[r]) \,\}|$$

Despite being directly linked to the memorization of a canary, the rank does not necessarily relate to the probability of a sequence generated using a greedy or beam search of the most likely suffixes [7]. However, as shown in [7], using more advanced Dijkstra-based search methods or a long enough prefix allows one to recover the top-ranked canaries.

The rank requires one to compute the log-perplexity of all possible canaries, making it a computationally expensive measure. For this reason, [7] introduced the

exposure: a quantity based on the rank but which can be efficiently approximated.

In particular, exposure measures how access to the model improves guesses about a secret such as that represented by an insert canary. In other words, exposure encodes the increase in information about an inserted canary gained by accessing the model. The paper provides a nice derivation of the exposure metric as the reduction in the guessing entropy[4] of $s[r]$ when the model f_Θ is available for query. Formally, exposure is defined as follows.

Definition 3.1.3 (Exposure)**.** Given a canary $s[r]$, a generative sequence model f_Θ with parameters Θ, and randomness space $\mathcal{R}$, the *exposure* of $s[r]$ is

$$\text{exposure}_\Theta(s[r]) = \log_2 |\mathcal{R}| - \log_2 \text{rank}_\Theta(s[r])$$

The exposure is a non-negative real value that ranges between 0 and $\log_2 |\mathcal{R}|$, and thus its magnitude depends on the size of the search space $\mathcal{R}$. In particular, the maximum value is achieved by the highest ranking and, therefore, most likely canary, while the minimum value is achieved by the least likely one. Therefore, if the inserted canary registers a high exposure, it indicates that the model memorized it quite extensively. In contrast, a low exposure would indicate that the inserted canary is just as likely as any other canary that could have been inserted.

3.1.2 Exposure-Based Testing in Practice

Here are summarized the most interesting findings of [7]. In particular, following the exposure-based testing methodology described in Section 3.1.1, it has been shown that the degree to which neural network based generative models are susceptible to unintended memorization may greatly vary from one model to another.

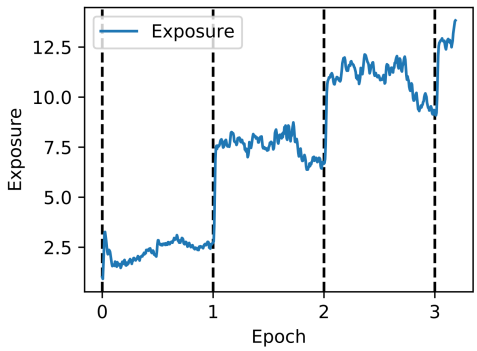
In [7], the exposure-based testing methodology is applied to Smart Compose, a word-level LM trained on the personal emails of millions of users and actively used by Gmail users for text completion. Smart Compose is an LSTM-based RNN with millions of parameters and trained on billions of word sequences utilizing a vocabulary of tens of thousands of words. To apply the testing procedure to Smart Compose, [7] used canaries of 5 and 7 randomly selected words, with the first two and last two words being known context. By inserting the canaries from 1 to 10k times in the training data, with both formats, the observed exposure signal was too feeble to allow for discovery using the extraction algorithm proposed in [7].

In addition to LMs, [7] applied the exposure-based testing to a neural machine

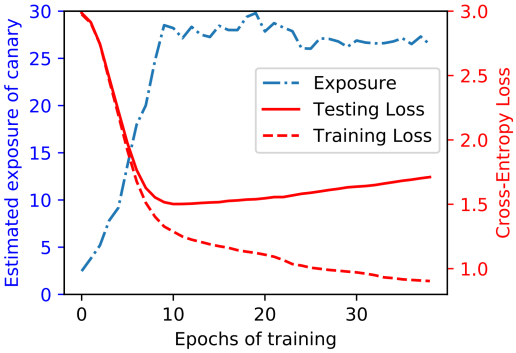
[4]The guessing entropy is defined as the expected number of guesses required in an optimal strategy to determine the value of a discrete random variable.

translation model from the TensorFlow model repository, which, given as input a sequence of words in English, outputs the corresponding sequence of words in Vietnamese. The canary used had the format "My social security number is □□□−□□−□□□" and was inserted into a training set of 100k pairs of sentences. Using an adaptation of the exposure metric for this task, [7] found that after just four canary insertions, the canary becomes completely memorized (i.e., the canary has maximum exposure) and can be easily extracted.

Another important finding of [7] is that memorization is part of learning, as it begins to occur during the first epoch of training (see Figure 3.1a). Moreover, the exposure increases until the minimum validation loss is reached, and after that, it no longer increases (see Figure 3.1b). Therefore, exposure increases when the model is learning and stops to increase when the model is not learning, suggesting that unintended memorization must be a necessary underlying component of training.



(a) Exposure of a single canary inserted at the start of the training set and shuffling disabled.



(b) Relation between exposure and validation loss during training of a model with limited training data to quickly overfit.

Figure 3.1: Unintended memorization during training [7]. Unintended memorization begins to occur during the first epochs of training and happens when the model is learning (i.e., the validation loss is decreasing).

In addition to assessing unintended memorization using the testing methodology based on artificially inserting canaries, [7] also studies unintended memorization with naturally occurring secrets already present in the training dataset. In particular, the Enron email dataset, consisting of 500k emails exchanged between the employees of the Enron Corporation, was used to extract secrets without inserting canaries. They partitioned emails by sender and trained a character-level LSTM-based LM for users who have sent at least a secret. Then, using their exposure-based extraction algorithm, [7] successfully extracted three secrets corresponding to two credit card numbers and one social security number.

3.1.3 Prevent Unintended Memorization

After establishing that unintended memorization occurs in neural networks, [7] also evaluates potential defenses against such memorization. In detail, three possible defenses were analyzed: regularization techniques, text sanitization, and differentially private SGD.

It was empirically shown that regularization approaches, such as weight decay, dropout, batch normalization, and quantization, do not mitigate the problem as no sensible reduction in exposure was observed. However, [7] found that the choice of architecture and hyperparameters affects memorization to some extent. Consequently, exposure can be used to make an informed decision when designing a model. For example, it could be another objective guiding the hyperparameter optimization by not only searching for the best performing model but also for the one that less exposes artificially inserted secrets. Or it can signal that specific mitigations are required if the model is trained on sensitive data.

Regarding sanitization, which is based on removing potentially sensitive text segments, [7] concludes that it cannot guarantee the removal of all secrets occurring in the training data. For example, [7] failed to remove all secrets by using an algorithm that compared the perplexities of two models trained on non-overlapping subsets of training data and removed the sentences for which the two models disagreed. However, sanitization, for example, based on the removal of blacklisted terms and patterns, is a best practice that should always be employed before processing sensitive or private data.

Differentially private SGD, which will be covered in Section 3.5.1, was found to eliminate memorization completely. However, it also comes with a price in terms of performance reduction, training time, and difficult application.

On a concluding note, it is important to note that exposure represents a *lower bound* to the actual amount of memorization that occurs in the model. On the other hand, differential privacy provides a rigorous *upper bound guarantee* of how much memorization could, in theory, occur. With exposure and differential privacy, the *actual* amount of memorization can thus be bounded from below and above. However, [7] empirically observed that applying differential privacy even with weak guarantees (which in theory would be meaningless) kills the exposure signal completely, making memorization not observable with the methodology they propose, suggesting that both bounds are probably loose.

3.2 Training Data Extraction from Language Models

In generative models (e.g., autoregressive LM), an attacker with access to the model can generate data and predict the membership of such data to the model training set. This is a form of the well-studied membership inference attack (MIA) (Section 3.3) applied to generative models. In particular, as we have seen (Section 3.1.2), it has been established that neural networks tend to memorize rare details of their training data. Therefore, if a generative model discloses some rare, sensitive information about an individual (or organization), this indeed represents a major privacy breach.

Carlini et al. [8] continued their effort to assess memorization in neural networks. In particular, they attack the autoregressive LM GPT-2 to extract individual training sequences by simply accessing the model in a black-box fashion. They focus on GPT-2 as it is a state-of-the-art public model trained on a large amount of data from the public Internet, which, however, was never released by OpenAI.

Figure 3.2 reports an example of a rare training sequence successfully recovered from GPT-2 that contains the redacted personally identifiable information of an individual (full name, physical address, email address, etc.).

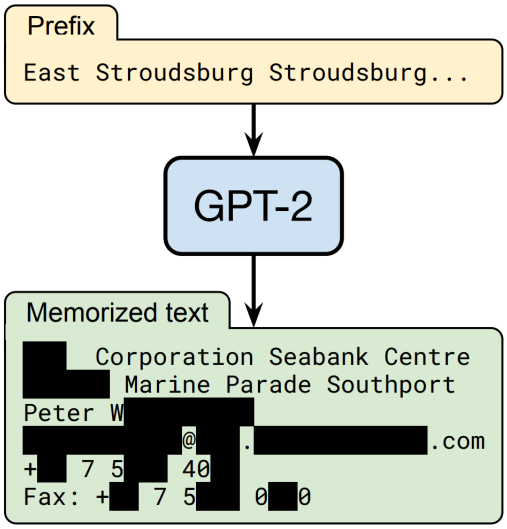


Figure 3.2: Training data extraction from a language model [8]. With black-box access to GPT-2, [8] managed to successfully extract memorized rare sequences from the training data. The extracted sequences included personally identifiable information of different individuals and long sequences with high entropy (UUIDs, hash codes, etc.).

The attack pipeline to extract memorized sequences proposed by [8] is quite straightforward:

1. Generate many sequences.

2. Infer the membership of the generated sequences to find possible sequences that belonged to the training data.

In [8] different strategies were used to generate candidate sequences. For example, after initializing the LM with a start-of-sentence token, they repeatedly sampled tokens with top-k strategy[5]. However, this simple strategy was observed to produce the same or similar sentences repeatedly, as it favors sequences that are probable from start to end.

To generate a more diverse set of sentences, [8] used two other strategies. One based on a temperature parameter[6] that decays over time from a high value to one: this ensures that, at the start of the sentence, the model explores a more diverse set of prefixes, and as the prediction progresses, it follows more high-confidence paths (i.e., those suffixed for which the model assigns higher probabilities). The last strategy consisted in prompting the model with short prefixes (of 5 to 10 tokens) scraped from the Web, as it is the same source as GPT-2 training data.

A basic approach to infer the membership of the generated sequences to the training data and to identify potentially memorized pieces of text is to use the perplexity measure (Definition 3.1.1). However, it was observed that many non-memorized sequences, such as sequences with repeated substrings (e.g., "I love you. I love you..."), were assigned low perplexity (i.e., high likelihood), so using just perplexity was considered ill-suited, as it provides many false positives.

To filter out uninteresting yet low-perplexity sequences, [8] used a relative approach that compares the perplexity of the model with that assigned by a second model since this second model will likely also assign low perplexity to these uninteresting sequences. Therefore, only those sequences for which the original model assigns *unexpectedly* high likelihood compared to a second model are considered. For example, they use this approach by using the ratio of the perplexity assigned by the largest GPT-2 XL model (1.5B parameters) with its smaller siblings: medium (345M) and small (117M). In addition, they also used the ratio between the GPT-2 perplexity and the zlib entropy (i.e., the number of bits of entropy when the sequence is compressed with zlib) and the ratio between the perplexities assigned by the model on the original sequence and its lowercase version.

[5]The top-k sampling strategy consists in setting all but the k largest probabilities to zero. The probabilities are then renormalized, to sum up to one before sampling.

[6]The temperature parameter T is used to rescale the vector z of raw non-normalized predictions (i.e., logits) before applying the softmax operation, that is $\text{softmax}(z/T)$. This rescale, for $T > 1$, has the effect of flattening the probability distribution, making the model less confident about its prediction.

After generating 600k (256 tokens long) sequences following the three generation strategies briefly described above, they selected 1.8k potentially memorized samples using the different membership inference metrics[7]. Then, with the help of the OpenAI researchers that trained GPT-2 and have access to its training data, they manually verified that of the 1.8k selected sequences, 604 were memorized sequences from the training data.

They recovered a wide and diverse set of memorized sequences, including personally identifiable information (names, addresses, phone numbers, etc.) of people appearing in only a few documents[8], URLs resolving to live Web pages, snippets of source code, high entropy sequences[9] (UUIDs, hashes, random IDs, etc.), but also content that has been since removed from the Web.

Using a training document containing multiple Reddit URLs appearing a varying number of times, [8] carried out an interesting experiment to correlate memorization with model size and insertion frequency. In particular, in the *single* training document, the most frequent URL appeared 359 times, while the least frequent URL appeared 8 times.

They provided the initial part of each URL, without the last part consisting of a 6-character ID and the title of the post, to the model and generated 10k extensions with top-k sampling. Then they checked whether the URLs emitted by the model appeared in the training document.

They found that GPT-2 XL emitted *all* URLs that appeared at least 33 times; GPT-2 medium emitted *some* URLs that appeared at least 56 times; GPT-2 small did not emit any of the URLs with this strategy. However, when the model was provided with a longer context (the 6-character ID) and beam search was used instead of top-k sampling, GPT-2 medium emitted *most* (all but one) of the URLs that appeared at least 35 times, while GPT-2 small emitted the most common URL that appeared 359 times.

These results are very informative, and a clear trend can be observed. As the model gets bigger and, as such has more capacity, the amount of memorized data increases. This result raises more concerns about data privacy, as the trend is to produce larger and larger models. Moreover, even if overfitting may be the cause of other privacy leakages, it is undoubtedly not the case for the state-of-the-art GPT-2 family of models, as these models are trained (by extremely competent researchers)

[7] They only selected 1.8k sequences because they were limited by the amount of work they could do to manually verify whether each of the selected sentences appeared in the training set.

[8] For example, they recovered the usernames of six users participating in an IRC conversation that appeared in just *one* training sample.

[9] For example, they recovered a sequence of 87 alphanumeric characters that appeared 10 times in only a *single* training document.

on massive datasets.

3.3 Membership Inference Attacks Against Classification Models

A byproduct of ML models memorizing their training data is that they can leak information about the individual data records on which they were trained. In particular, a key observation is that ML models behave differently on data on which they were trained versus data they see for the first time [5]. For example, in a classification task, a given model will generally have greater confidence in a sample seen during training compared to a sample seen for the first time while deployed. A prominent example of this behavior is when a model overfits its training data. This observation can be used to carry out what is called a membership inference attack.

The membership inference attack (MIA) [5], [6] is a type of attack in which an attacker has access to a model and wants to determine whether a given data sample was part of the training data. The access to the target model can be a simple gray-box access, i.e., the model outputs the prediction vector with the probability for each class. Figure 3.3 illustrates the membership inference attack against a classification model.

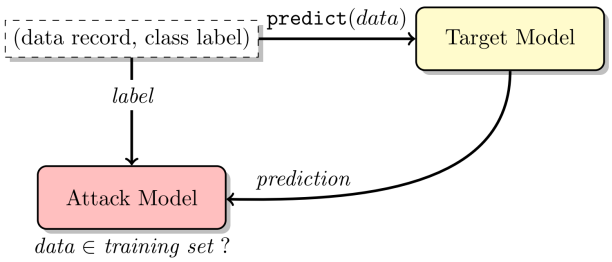


Figure 3.3: Membership inference attack against a classification model [5]. The attacker queries the target model with a target sample and obtains the prediction vector, which contains for each class the probability that the sample belongs to that class. The target sample and the prediction vector are passed to the attack model, which infers whether the target sample was in the training set of the target model.

As a motivating example of MIA aimed at undermining user privacy, imagine that a model, used to determine the correct dose of a certain drug for some disease, is trained on the clinical logs of patients subjected to the disease. The attacker has the clinical record of John Doe and wants to determine if it was used to train the model, to discover whether John Doe has the associated disease.

From the definition of the MIA and the above example, we see that MIA represents a privacy threat if all the following conditions are met:

- The attacker has access to the trained model;
- The attacker has access to data that may have been used in the training of the model;
- The discovery by the attacker that some data were used in training represents a privacy threat for the data owner.

If one of the above conditions is not met, we can conclude that MIA does not represent a concrete privacy threat.

As mentioned above, the main intuition behind MIA is that models tend to behave differently on their training data compared to novel data not seen during training. The attack model aims to learn to distinguish this discrepancy in behavior between the target model train and test data. To achieve this goal, the attacker does not use the target model, but they are required to access some *shadow data*, that is, disjoint data[10] drawn from the same underlying distribution of the target model training data. The shadow data are divided into train and test splits, and the train split is used to train various *shadow models* that may have partially overlapped training datasets. The shadow models should have similar architectures and be trained analogously to the target model so that the target model and the shadow models behave similarly. The attack model learns to distinguish between behaviors using these shadow models. In particular, the attack model is trained to discriminate between the data samples being *in* and *out* of the target model training data using the predictions the shadow models produce on their shadow train and test data, as shown in Figure 3.4.

To be more precise, the attack model consists of multiple attack models, one for each output class of the target model, trained as described. This is done because, depending on the true class of the input sample, the target model produces different distributions over its output classes. Therefore, an attack model for each class will generally achieve higher accuracy.

What has been described is the adversarial use of ML models to carry out an MIA as described in [5], where they focus on MIA against target models trained by ML-as-a-Service providers (e.g., Google Vertex AI, Microsoft Azure, Amazon ML).

In the literature, different variations of this attack have been proposed. For example, [6] focuses on NLP classification models using sample-level (as the one just

[10]Disjoint shadow data is the worst case for the attacker. MIA will perform even better if the attacker access shadow data that overlap with the target model training data.

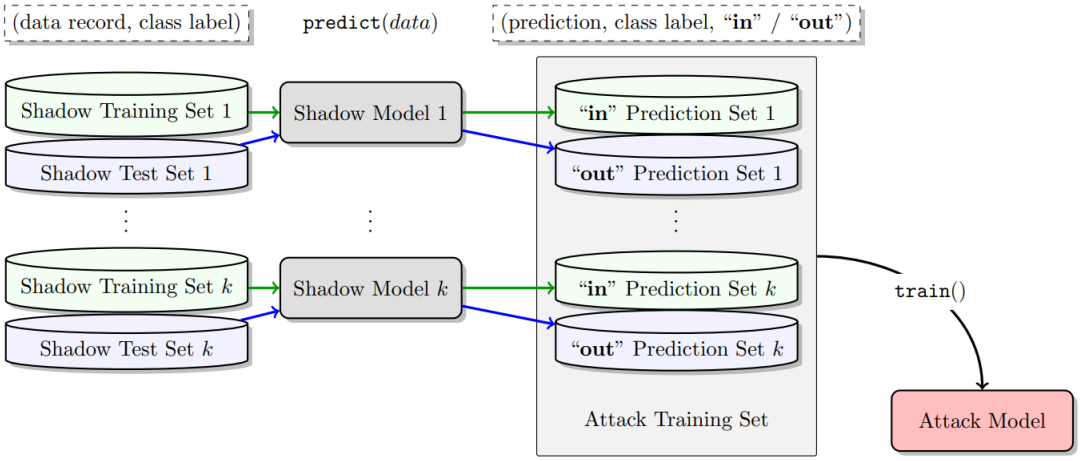


Figure 3.4: Training of a membership inference attack model [5]. The predictions produced by the shadow models on the train shadow data (labeled with *in*) and on the disjoint test shadow data (labeled with *out*) are used to train the attack model. The attack model learns to discriminate the behavior of the shadow models on their training data with respect to their testing data.

described) and user-level membership inference attacks. However, unlike sample-level MIA, which aims to infer the membership of a single sample, user-level MIA seeks to infer the membership of user data by combining multiple samples from the same user and, therefore, can be more accurate. In particular, to implement a sample-level MIA, [6] employs a simple threshold-based strategy that we briefly describe here.

Let M_Θ be the target model trained on the training dataset $\mathcal{D}$ containing C classes and denote by $\mathcal{D}_c$ its subset limited to samples from class $c \in \{0, \dots, C-1\}$. Given a labeled sample (x, y), M_Θ maps the features vector x into a vector of probabilities $p = M_\Theta(x) \in \mathbb{R}^C$.

Moreover, let $\bar{\mathcal{D}} \cup \bar{\mathcal{D}}'$ ($\bar{\mathcal{D}} \cap \bar{\mathcal{D}}' = \emptyset$) be the shadow data from the same distribution of $\mathcal{D}$, where $\bar{\mathcal{D}}$ is used to train a shadow model $M_{\bar{\Theta}}$ and $\bar{\mathcal{D}}'$ is not used in training $M_{\bar{\Theta}}$.

Let $F : \mathbb{R}^{C+1} \to \mathbb{R}$ be the function that, given the prediction vector of probabilities $p \in \mathbb{R}^C$ and the ground truth label $c \in \{0, \dots, C-1\}$, computes a feature that can be used to discriminate between a member and non-member samples of the training data. Different features can be used. For example, some of the features used in [6] are:

- Confidence of the model

$$F(p, c) = p[c]$$

- Rank of the correct label

$$F(p, c) = |\{\, i \mid p[i] \geq p[c] \,\}|$$

- Modified entropy

$$F(p, c) = -(1 - p[c]) \log p[c] - \sum_{i \neq c} p[i] \log p[i]$$

For each class $c \in \{0, \ldots, C-1\}$, the discriminating features for the samples in $\bar{\mathcal{D}}_c$ and $\bar{\mathcal{D}}'_c$ are computed using the function $G(x, y) := F(M_{\bar{\Theta}}(x), y)$. In particular, the set of features $\mathcal{F}_c$ for the (member) samples in $\bar{\mathcal{D}}_c$ is computed as

$$\mathcal{F}_c = \{G(x, y) \mid (x, y) \in \bar{\mathcal{D}}_c\}$$

Similarly, the set of features $\mathcal{F}'_c$ for the (non-member) samples in $\bar{\mathcal{D}}'_c$ is computed as

$$\mathcal{F}'_c = \{G(x', y') \mid (x', y') \in \bar{\mathcal{D}}'_c\}$$

The threshold of class c is calculated to best separate the features of shadow members and non-members, that is

$$\tau_c = \underset{\tau \in \mathbb{R}}{\operatorname{argmax}} \left(\sum_{f \in \mathcal{F}_c} \mathbb{1}[f \geq \tau] + \sum_{f' \in \mathcal{F}'_c} \mathbb{1}[f' < \tau] \right)$$

Finally, given the target data sample (x_t, y_t), the threshold value τ_{y_t} associated with the class y_t is used to infer whether the sample was a member of the target model training data $\mathcal{D}$ according to

$$\mathbb{1}[F(M_{\Theta}(x_t), y_t) \geq \tau_{y_t}]$$

Despite its simplicity, this threshold-based MIA can be quite effective and perform as well as more sophisticated ones based on neural networks [6].

A user may contribute with multiple samples to the target model training data, and the attacker can use such samples to determine the membership of the user's data to the training set. In [6], different strategies are applied to carry out this user-level MIA. For example, building upon the above threshold-based MIA, a straightforward and effective extension could be to determine the membership of each sample in the target user's data (using the class-wise thresholds τ_c obtained as before) and infer if the user data were part of training by majority voting over each sample's

estimated membership. An alternative and more involved approach would consist in generating a vector of aggregate metrics (e.g., average, variance, minimum, maximum) over the discriminating features of all the user's samples labeled with the corresponding membership (i.e., *in* or *out* the training shadow data) and fitting a classifier (e.g., logistic regressor) to estimate the membership. However, this last approach does not exploit the susceptibility of each class, as different classes can experience different generalization errors, and such errors strongly correlate with MIA success.

3.4 Attacks Against Embedding Models

Embedding models are, in general, functions that map raw inputs — such as texts, nodes in a network, or geographic locations — to dense vectors of numbers. Embeddings are usually used in conjunction with *transfer learning*, in which large amounts of unlabeled data are used to train embedding models, and the representations they produce can be later transferred to various downstream tasks. Transfer learning is especially useful when the labeled data for the downstream task are limited or difficult to collect.

In the NLP domain, it has recently been shown [9], [10] that text embeddings can leak sensitive information about the inputs in addition to encoding the generic semantics of input texts.

Song and Raghunathan [9] proposed and studied a taxonomy of attacks against sentence embeddings illustrated in Figure 3.5. In particular, they identify three types of attacks.

- In *embedding inversion*, the attacker aims to reconstruct the words in the input text given its embedding.
- In *attribute inference*, the attacker aims to infer sensitive attributes, such as the authorship of the text, from the embedding.
- In *membership inference*, the attacker wishes to infer if a particular *context* appears in the training data of the embedding model. The definition of context depends on the embedding model: for word embeddings, the context is a sliding window of words, while for sentence embeddings, the context is a pair (or more) of sentences. Membership inference is directly related to the memorization that occurs during model training.

Note that membership inference concerns training data privacy, whereas embedding inversion and attribute inference concern inference-time input privacy.

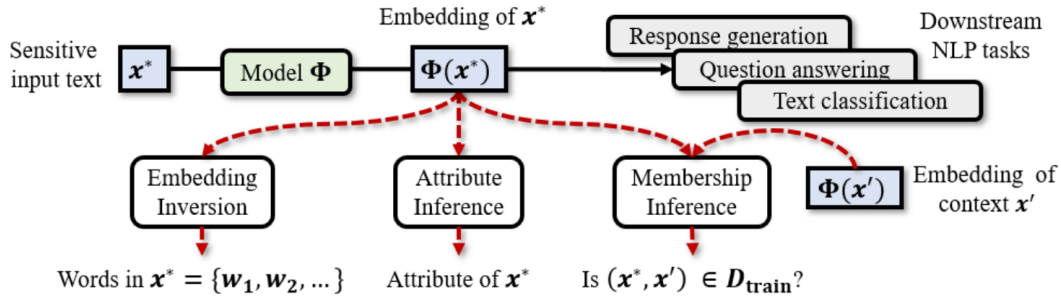


Figure 3.5: Taxonomy of attacks against embeddings [9]. Given the embedding $\Phi(x^*)$ of a sensitive input text x^* produced using the model $\Phi(\cdot)$, an attacker may attempt to: (1) invert the embedding to find the words appearing in x^*, (2) infer sensitive attributes of x^*, (3) infer if x^* and a possible context x' has been used for training $\Phi(\cdot)$, i.e., membership inference attack.

Given the nature of embedding models, which are usually trained on large amounts of *publicly* available corpora, inversion and attribute inference are more interesting attacks and are therefore covered in the following sections.

Pan et al. [10] empirically confirmed the leakage of sensitive information when using embedding models. In particular, they proposed two classes of attacks against embeddings from different state-of-the-art pre-trained LM, including BERT and GPT-2. In *pattern reconstruction attack*, the attacker knows the underlying format of the text (e.g., genome sequence) and uses this knowledge to infer some sensitive information from the embedding. In *keyword inference*, the attacker aims to establish whether some keyword appears in a text by accessing the corresponding embedding (e.g., what part of the body is mentioned in the embedding of a medical description).

However, since [9] is broader in scope, for example, by not making strong assumptions about the structure of the text, here we focus on the main results provided there. In particular, Section 3.4.1 explains the white-box sentence embeddings inversion attack, Section 3.4.1 provides an example of sensitive attribute inference from sentence embeddings, and finally, Section 3.4.3 explains adversarial training as possible mitigation applied to limit inversion and attribute inference attacks.

3.4.1 Sentence Embeddings Inversion

In [9], two variants of the inversion attack against sentence embeddings are proposed. In the white-box variant, the attacker has access to the model and its internal weights, whereas, in the black-box one, the attacker can only query the embedding model without accessing its weights. Since Chapter 4 focuses on a variation directly derived from the black-box proposed in [9], here we just briefly describe the white-

box variant.

The idea behind embedding inversion is to find the sequence of words (which make up a sentence) whose embedding has the minimum ℓ_2 distance from the target embedding $\Phi(x^*)$ of a text x^* computed by the embedding model $\Phi(\cdot)$.

A naive approach to the inversion task would be to enumerate all possible word sequences and query the embedding model to produce the corresponding vectors to find the sequence that has the minimum distance. However, given a vocabulary $\mathcal{V}$ and sentence length L, the space of all possible combinations has cardinality $|\mathcal{V}|^L$. Consequently, this brute-force approach would be computationally infeasible for values of L that are not trivially small.

The discrete space of word combinations can be relaxed to a continuous space. That is, each position in the input sentence can be assigned a probability vector over $\mathcal{V}$, reporting for each word the probability of appearing at that position.

More precisely, using continuous relaxation, each word position $i = 1, \ldots, L$ in the sentence is assigned a vector $z_i \in \mathbb{R}^{|\mathcal{V}|}$, and all the z_i vectors are grouped into the matrix

$$Z = \begin{bmatrix} z_1^\intercal \\ \vdots \\ z_L^\intercal \end{bmatrix} \in \mathbb{R}^{L \times |\mathcal{V}|}$$

Correspondingly, the *soft* words at each position i are computed as

$$\hat{v}_i = V^\intercal \cdot \text{softmax}(z_i / T) \in \mathbb{R}^d$$

where $V \in \mathbb{R}^{|\mathcal{V}| \times d}$ is the matrix of word embeddings[11] with an embedding for each row, and T is the temperature hyperparameter. The sequence of all the soft words, grouped into the matrix

$$\text{relaxed}(Z, T) = \begin{bmatrix} \hat{v}_1^\intercal \\ \vdots \\ \hat{v}_L^\intercal \end{bmatrix} \in \mathbb{R}^{L \times d}$$

The embedding model $\Phi(\cdot)$ is then used to compute the corresponding sentence embedding. The inversion problem can then be expressed as the following minimization problem

$$\min_Z \| \Phi(\text{relaxed}(Z, T)) - \Phi(x^*) \|_2^2 \tag{3.1}$$

[11] A sentence embedding model, as a first step, usually maps a sequence of words into a sequence of word vectors using a word embedding matrix. The word embedding matrix is, therefore an internal weight matrix accessible in a white-box setting.

which can be solved using gradient-based optimization methods since each operation is continuous, and gradients can be computed with back-propagation thanks to access to the internal weights. Finally, the text is obtained by selecting the most probable word at each position as

$$\hat{x} = \{w_i \mid i = \arg\max z_j\}_{j=1}^{L}$$

The relaxation approach of Equation 3.1 was empirically observed to have good performance when used against embeddings produced by a dual-encoder[12] sentence embedding model implemented with an LSTM or three stacked Transformer encoder layers. However, it yielded very poor performance when used against the embedding produced by mean pooling over the last hidden representations of a deep pre-trained LM, such as BERT or ALBERT. A possible explanation is that the embedding produced after many processing layers is more abstract. Therefore, semantically similar sentences are assigned to very close points in the embedding space, making the inversion much harder.

To mitigate the above problem, a model M (e.g., least squares regressor) can be trained, using auxiliary data, to map the output embedding to a lower-layer representation. In particular, the lowest possible representation considered in [9] is the average of the embeddings (from V) of the words appearing in the sentence. Then, by defining the vector $z \in \mathbb{R}_{\geq 0}^{|\mathcal{V}|}$ that contains the contribution of each word to the average, the white-box inversion attack can be expressed as the following minimization problem

$$\min_{z} \| V^{\intercal} \cdot z - M(\Phi(x^*)) \|_2^2 + \lambda \|z\|_1 \tag{3.2}$$

where, again, V is the matrix of word embeddings, while λ represents the constraint to induce a sparse vector z. The optimization problem can be solved with projected gradient descent, which, after each descent step, sets z_j to 0 if $z_j < 0$ [9]. Finally, the recovered words are obtained by selecting those with a coefficient z_j not smaller than a sparsity threshold hyperparameter τ as

$$\hat{x} = \{w_i \mid z_i \geq \tau\}_{i=1}^{|\mathcal{V}|}$$

Using the relaxed inversion approach of Equation 3.1, [9] managed to recover around 57% of the words from the embeddings produced by an LSTM-based dual-

[12]The dual-encoder model is trained on pair of sentences in context (e.g., question and answer) using the contrastive learning framework.

encoder and 36% using the Transformer based one. However, as mentioned above, when used against the embeddings obtained from BERT and ALBERT models, performance decreased drastically: only less than 1% and around 3% recovered words from the BERT and ALBERT representations, respectively.

With the sparse coding formulation of Equation 3.2, performance improved across the board with around 60%, 63%, 50%, and 62% recovered words from the LSTM, Transformer, BERT, ALBERT, respectively. Furthermore, it was observed that using data from a different domain to train M has little impact on overall performance (53%, 57%, 45%, 60%).

3.4.2 Sensitive Attribute Inference

Quality embeddings can capture rich semantic information about the input data. In particular, embeddings may be much more informative than the raw input itself, and this is mainly the reason they work very well in practice. Consequently, when embeddings are produced on users' data, such as their watch history or personal messages, embeddings may naturally encode personal information about the users themselves.

The fact that embeddings can encode such rich information can be exploited by an attacker who intends to reveal sensitive attributes that might not even appear in or be easy to infer from the raw input. In this setup, the attacker must have access to a (realistically limited) dataset of embeddings labeled with the corresponding sensitive attribute. This dataset can then be used to train a simple classifier that assigns the attribute to the given embedding.

To demonstrate that this information leak is possible in practice and represents a concrete threat, [9] considers the scenario in which the sensitive attribute is the author of a text. Therefore, the attacker's objective is to de-anonymize the author by just accessing the embedding of a text. Toward this purpose, different embedding models were analyzed in [9]: the LSTM and Transfomer dual-encoder models used in the inversion attack (Section 3.4.1), Skip-thought and InferSent. The results of authorship inference for the said models, including a baseline model (TextCNN), trained on the raw sentences rather than the embeddings, from [9] are reported in Figure 3.6.

Interestingly, with only a few labeled examples per author (e.g., $N_s = 10$), the top-5 accuracy in classifying authors is relatively high and naturally increases with more examples. Another interesting finding of [9] is that the nature of the training objective of the embedding model strongly influences the leakage of authorship information.

In particular, the dual-encoder models (LSTM and Transfomer based) trained

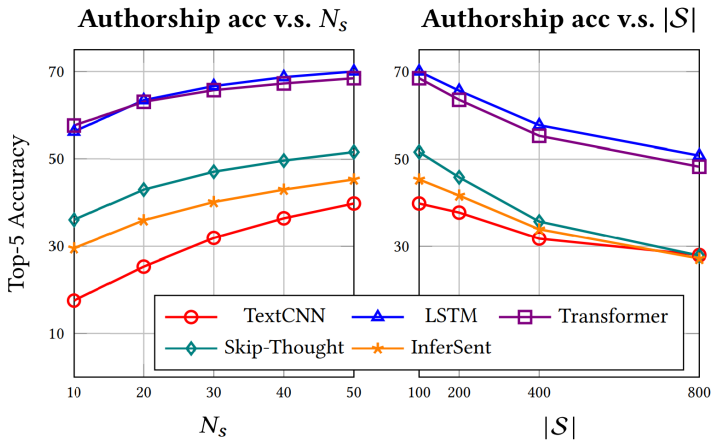


Figure 3.6: Results of authorship inference from embeddings [9]. The left figure reports the top-5 accuracy in classifying $|\mathcal{S}| = 100$ authors as a function of the number of labeled examples per author. The right figure reports the top-5 accuracy as a function of the number of authors $|\mathcal{S}|$ when the attacker has $N_s = 50$ labeled examples per author.

using a contrastive learning objective are the ones that leak the most this kind of information. On the other hand, Skip-thought, which is trained in an unsupervised fashion to generate the context given a sentence, and InferSent, which is trained on supervised tasks, are less prone to this leak. Moreover, as mentioned earlier, all the embeddings capture more information than the raw input, since TextCNN is the least performing model across all settings.

A possible explanation for why the contrasting learning framework favors this type of leakage is that embeddings are learned to be close to data that appear in context in the training set and far from negative samples. In particular, in unsupervised frameworks, the training data of the embedding model are assigned to latent classes. In [9], the embedding models are trained on sentences from the BookCorpus dataset[13]. Therefore, given that all the sentences in context come from the same author, it is reasonable to assume that a possible latent class learned by the model could be the author of the book from which the sentence is taken. Thus, even a simple classifier with a few training examples can easily learn to discriminate between embeddings generated from texts by different authors.

[13]Additionally, the sentences used to assess the authorship inference accuracy are taken from held out data from the same BookCorpus dataset. This is undoubtedly the best-case scenario for the attacker, as the authors are also present in the training of the embedding models.

3.4.3 A Possible Mitigation: Adversarial Training

A possible first line of defense against inference-time attacks, such as embedding inversion and attribute inference, is adversarial training. In adversarial training, a model is jointly trained to optimize a primary objective and, simultaneously, to minimize the utility of some simulated adversary $\mathcal{A}$. For example, in [9], adversarial training is applied to the dual-encoder sentence embedding model to mitigate both inversion and attribute inference.

In the case of inversion, $\mathcal{A}$ is trained to infer the set of words $\mathcal{W}(x)$ in x given $\Phi(x)$, while Φ is trained to maximize the similarity of sentences in context. Therefore, given a pair of sentences in context (x_a, x_b) and a set of negative sentences $\mathcal{X}_{\text{neg}}$, the learning objective becomes the following minimax optimization problem

$$\min_{\Phi} \max_{\mathcal{A}} \underbrace{-\log P_{\Phi}(x_b|x_a, \mathcal{X}_{\text{neg}})}_{\text{primary objective}} + \lambda \underbrace{\log P_{\mathcal{A}}(\mathcal{W}(x_b)|\Phi(x_b))}_{\text{leaked information}} \tag{3.3}$$

in which the term λ controls the trade-off between learning a model producing good representations and a model that leaks the least amount of information to an adversary. In particular, a possible choice of the loss to optimize $\log P_{\mathcal{A}}$ in Equation 3.3 is

$$\mathcal{L}_{\mathcal{A}} = -\sum_{w \in \mathcal{V}} y_w \log(\hat{y}_w) + (1 - y_w) \log(1 - \hat{y}_w)$$

where $y_w = \mathbb{1}[w \in \mathcal{W}(x)]$ and $\hat{y}_w = P(y_w|\Phi(x))$ is the probability, estimated by the simulated adversary, that the word y_w appears in x.

To mitigate attribute inference, Equation 3.3 can be adapted by substituting $P_{\mathcal{A}}(\mathcal{W}(x_b)|\Phi(x_b))$ with $P_{\mathcal{A}}(s|\Phi(x_b))$.

In [9], the use of adversarial training is especially beneficial for the attribute inference attack since the attacker's performance drops consistently while the performance on different downstream benchmarks stays competitive. On the other hand, against inversion, depending on the model used, the downstream performance may drop noticeably. At the same time, the attacker can, in any case, still recover a decent amount of words from the embeddings.

Adversarial training only mitigates the attack to some extent. It may also happen that novel, more performing attacks get devised, and a model trained with this strategy may not be as robust. Moreover, it does not provide any formal provable guarantee such as those offered by differentially private mechanisms.

3.5 Differential Privacy

In data analysis, many attempts have been made to define privacy in a practical way [29]. A naive approach to data privacy is *anonymization*, which consists in removing identifying information from data (e.g., the person's name from a medical record). However, this approach may be insufficient to protect privacy: the nonremoved information can still be used in *linkage attacks* to identify a person by using other available sources of information. Multiple successful linkage attacks have been performed on anonymized datasets. For example, researchers managed to de-anonymize the Netflix Prize dataset containing the anonymous movie ratings of its subscribers by linking the information with the Internet Movie Database (IMDb) [30].

Another approach to privacy that aims to limit linkage attacks is *k-anonymity*. A dataset is said to be k-anonymous if for any record associated with a person in the dataset, there are at least $k - 1$ other records indistinguishable from it. Still, k-anonymity has its shortcomings because it requires a large enough dataset and a small number of features for each record.

The modern approach to privacy is represented by *differential privacy* (DP) [31], [29], [32] which provides a formal provable privacy guarantee that applies to data analysis algorithms.

The intuitive idea behind privacy, as provided by DP, is that if a person decides to participate in a dataset, they are assured that their data will have a negligible impact on them in the future. To give a concrete example, assume that John Doe is asked if his data can be included in the training set of a machine learning model that computes the life insurance cost. Unfortunately, John suffers from some health condition that makes his life insurance cost higher. DP aims to guarantee that the inclusion of John Doe's information in the dataset will not affect him in the future by revealing his health conditions to someone.

According to the Fundamental Law of Information Recovery [31], with enough queries, the information contained in a dataset can be fully recovered, destroying any form of privacy. DP attempts to limit the amount of such leaked information when answering queries. In particular, DP is based on the concept of *randomized responses*.

Randomized responses add randomization when answering a query to provide *plausible deniability*. For example, in a survey asking students if they cheated during an exam, students could use a coin flip to decide whether to respond honestly or not. From the aggregate responses, it is still possible to obtain an estimate of the students who cheated. Still, at the same time, students have plausible deniability:

the answer they gave may or may not be the correct one, and thus they keep their privacy.

The probability of a correct response (50% in the example just described) controls the privacy protection provided: a smaller probability will yield noisier results while providing more privacy, whereas higher probabilities will provide less privacy protection but more accurate responses. In other words, the probability of a correct response controls the trade-off between the utility perceived by the questioner and the privacy preserved by the questioned.

DP offers protection against a very strong threat model [33]: it is *robust* against powerful adversaries, even when they have unbounded computation and access to arbitrary side knowledge; it does not rely on obscurity so that the algorithms can be revealed *publicly*. Finally, DP gives a rigorous way to *quantify* the amount of privacy that a mechanism provides.

The formal definition of differential privacy states the following.

Definition 3.5.1 (Differential Privacy)**.** Let $\epsilon > 0$, $\delta \in [0, 1]$, and $M : \mathcal{X} \to \mathcal{Y}$ a randomized algorithm. M is (ϵ, δ)-*differentialy private* if for any two datasets $\mathcal{D}, \mathcal{D}' \subset \mathcal{X}$ that differ in one element, it holds

$$\forall Y \subset \mathcal{Y},\ P[M(\mathcal{D}) \in Y] \leq \exp(\epsilon) \cdot P[M(\mathcal{D}') \in Y] + \delta$$

Notice that if we set $\delta = 0$[14], we can express the condition in Definition 3.5.1 as

$$\forall Y \subset \mathcal{Y},\ \log \frac{P[M(\mathcal{D}) \in Y]}{P[M(\mathcal{D}') \in Y]} \leq \epsilon$$

Intuitively, the definition of DP requires that, given two datasets $\mathcal{D}$, $\mathcal{D}'$ that differ by only one element (i.e., adjacent datasets), and a mechanism $M(\cdot)$ that operates on such datasets, the results of $M(\mathcal{D})$ and $M(\mathcal{D}')$ are almost indistinguishable for every choice of $\mathcal{D}$ and $\mathcal{D}'$.

Due to the randomized nature of the mechanism $M(\cdot)$, there exists a distribution over its possible outputs. In particular, Definition 3.5.1 requires that the divergence between the distributions over the mechanism output when applied to two adjacent datasets, is upper bounded by ϵ. The probability that this upper bound holds is controlled by the δ parameter: for $\delta = 0$, the upper bound must always hold. Figure 3.7 illustrates this intuitive idea in the case where the two datasets differ on John Doe's data.

The strength of this guarantee is controlled by the parameters ϵ and δ. ϵ bounds

[14]This corresponds to a stricter definition of differential privacy used in the literature.

how much the two output distributions of the mechanism can differ when two adjacent datasets are provided, and δ is a small probability that allows rare violations of this bound. The privacy guarantee becomes stronger as both parameters become smaller: to obtain meaningful privacy, ϵ should be a small constant (e.g., $\epsilon = 4$) while δ should be smaller than $1/|\mathcal{D}|$.

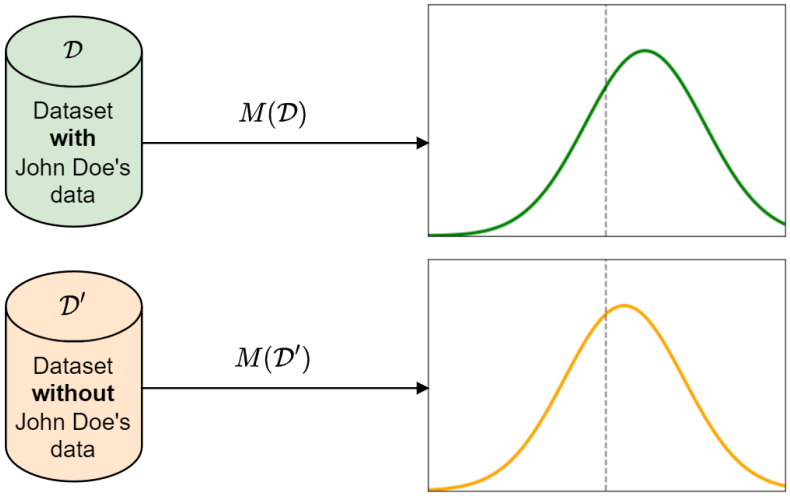


Figure 3.7: Intuition of a differentially private mechanism. If the mechanism is differentially private, the distributions over the mechanism outputs have a small divergence, and John's data is protected.

The mechanism M can be any operation that returns an output that depends on the information contained in the input dataset. For example, it could be an aggregate statistic (e.g., the number of students who passed the exam) or an algorithm that operates on the dataset (e.g., the training algorithm of a machine learning model).

DP mechanisms satisfy two essential properties: post-processing and composition. The post-processing property states that any function applied to the output of a DP mechanism will still result in a DP mechanism. According to the composition property, the composition of multiple DP mechanisms results in the overall mechanism being DP but with an increased privacy loss that depends on the guarantees provided by each composed mechanism.

A common approach to obtain a differentially private mechanism from a deterministic function $f : \mathcal{X} \rightarrow \mathcal{Y}$ is by adding controlled noise calibrated to the sensitivity S_f of the function, defined as

$$S_f = \max_{\mathcal{D},\mathcal{D}'} \|f(\mathcal{D}) - f(\mathcal{D}')\|$$

where $\mathcal{D}$, $\mathcal{D}'$ are adjacent datasets and $\|\cdot\|$ is either ℓ_1 or ℓ_2 norm.

For example, the Gaussian mechanism is defined as

$$M(\mathcal{D}) = f(\mathcal{D}) + \mathcal{N}(0, S_f^2 \cdot \sigma^2)$$

where S_f is computed using the ℓ_2 norm and $\mathcal{N}(0, S_f^2 \cdot \sigma^2)$ is the normal distribution with zero mean and standard deviation $S_f\sigma$. Roughly speaking, the larger σ, the more privacy-preserving the mechanism is.

3.5.1 Differentially Private Training of Neural Networks

As we have seen, neural networks tend to memorize their training data. Therefore, differential privacy can be used to build private and confidential models when trained on sensitive data. For example, companies such as Google [34] and Microsoft [35] are actively using DP to provide privacy to their customers. Some examples of DP applied to commercially deployed ML models include text completion assistants used in Microsoft Word, Outlook and Google Gboard.

The learning algorithm that maps a training dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ to a vector of weights of the neural network $\Theta \in \mathcal{Y} = \mathbb{R}^p$ can be made differentially private by using the Gaussian mechanism. In particular, the Gaussian mechanism can be applied to the classical non-private stochastic gradient descent (SGD) learning algorithm.

SGD, at each iteration, step t, samples a batch of examples $\mathcal{B}_t$ from the training dataset $\mathcal{D}$ and optimizes the weights Θ according to the update rule

$$\Theta^{(t+1)} = \Theta^{(t)} - \eta_t \frac{1}{|\mathcal{B}_t|} \sum_{i \in \mathcal{B}_t} \nabla \ell(\Theta^{(t)}) \tag{3.4}$$

where η_t is the step size at iteration t and $\ell(\cdot)$ is the sample loss function.

Two changes are required to obtain differentially private SGD [36], [37] or DP-SGD in short: (1) The gradients of each batch sample must be clipped to a maximum ℓ_2 norm of C and (2) Gaussian noise, with a standard deviation proportional to C, must be added to the average value of the clipped gradients. Intuitively, these changes are required to limit the influence of any particular sample on the weights updates. Therefore, a *trusted party* that trains the model on a private dataset using DP-SGD can publicly release the model (weights).

By defining the clipping function $\texttt{clip}_C : v \in \mathbb{R}^p \mapsto v \cdot \min\left\{1, \frac{C}{\|v\|_2}\right\} \in \mathbb{R}^p$, the

update rule of DP-SGD can be written as

$$\Theta^{(t+1)} = \Theta^{(t)} - \eta_t \left\{ \frac{1}{|\mathcal{B}_t|} \sum_{i \in \mathcal{B}_t} \frac{1}{C} \texttt{clip}_C \left[\nabla \ell(\Theta^{(t)}) \right] + \frac{\sigma C}{|\mathcal{B}_t|} \xi \right\} \tag{3.5}$$

where $\xi \sim \mathcal{N}(0, I_p)$ is a standard multivariate Gaussian random variable and σ is the standard deviation of the added noise [33].

The theoretical guarantees, defined by (ϵ, δ), provided by DP-SGD, are determined by the standard deviation σ, the batch size, and the number of total iterations. More in detail, each weight update leaks some information, and the overall privacy loss will increase at each step. In addition, using a larger batch size also increases privacy loss. To compensate for this loss, increasing the amount of added noise is possible. However, in practice, it is customary to fix some privacy budget (ϵ, δ), and the parameters that affect total privacy loss are determined as a consequence.

Moreover, the modifications brought by DP-SGD can create very different dynamics, making carefully chosen hyperparameter configurations and regularization settings for SGD unsuitable for its DP version.

The most popular deep learning frameworks include an implementation of DP-SGD or similar DP-based privatization techniques applied to other optimization techniques such as Adam. For example, Google released TensorFlow Privacy[15], and Meta AI released Opacus for PyTorch[16].

Unfortunately, there are some disadvantages to applying DP-SGD. The most obvious is a general decrease in the accuracy achieved by the model. This is because, as mentioned, gradients are clipped, and noise is added. Still, at the same time, the maximum number of update steps is limited by the privacy budget (ϵ, δ) defined a priori.

A possible approach to mitigate the reduction in accuracy could be to pre-train the model with public data and then use parameter-efficient fine-tuning strategies on the private data with DP-SGD [38]. However, no one-size-fits-all approach works for all tasks [39].

Moreover, even though DP-SGD assures the privacy (under the defined budget) of a single sample, a user may contribute to the training data with multiple samples [8], thus making the user's privacy not guaranteed under the same budget. For example, the credit card number of (the careless) John Doe may appear multiple times in the training data of an LM. Therefore, adaptations may be required. However, particular care must be taken when dealing with differential privacy to avoid

[15] `https://www.tensorflow.org/responsible_ai/privacy/`
[16] `https://opacus.ai/`

claiming that a mechanism is DP when it is not due to an error in a proof [40].

3.5.2 Differentially Private Text Representations

There is a multitude of other definitions and applications of differential privacy. What was described above falls under the umbrella of *centralized* DP, where there is a trusted party (e.g., who trains the ML model using DP-SGD) that accesses the sensitive data. Another flavor of DP is *local* DP, where the user applies a differentially private mechanism directly to their data. The aggregator that accesses the data is not trusted and can only access the anonymized data. For example, DP can be applied to obfuscate private information directly from text embeddings [41], or it can be used to directly sanitize text of sensitive information [42]. These or similar strategies could be considered as countermeasures to mitigate the leakage of information from vectorized texts as analyzed in Chapter 4.

Chapter 4

Experimental Methodology

This chapter describes the experimental methodology followed to validate the text representation models reviewed in Chapter 2 and to analyze the information leaked by the vectors such models produce. In particular, Section 4.1 provides the approach followed for the head-to-head comparison of the models in a downstream classification task; Section 4.2 describes the inversion attack mounted against the various models considered; finally, Section 4.3 outlines the methodology followed to implement the attack described in Section 4.2.

4.1 Benchmark Text Representation Models

This section describes the approach followed to compare the various representation models described in Chapter 2 using a downstream classification task. The objective is to ensure that the models considered in the information leakage attack described in Section 4.2 are effective and work as intended. Furthermore, the results of this benchmark can be used to relate the effectiveness of each representation in the downstream task with the amount of information that an attacker can recover from the representations.

4.1.1 Classification Dataset

The dataset used to perform the benchmark is a collection of news taken from an Italian newspaper, each labeled with its category (e.g., Politics, Culture, Sport). Therefore, the models used to produce the vectorized representation of texts for the classification task are fitted or trained on Italian texts. In particular, the Sentence-BERT model fine-tunes a BERT model pre-trained on an Italian corpus.

The dataset, which consists of around 150k news and ten possible labels, is not

entirely used. Given the unbalanced nature of the dataset and the simplicity of the downstream models considered, only eight possible classes with 2000 samples per class are used, resulting in a total of 16k samples balanced across eight classes. We made this decision because the objective is not to train the best model to categorize news articles by their content or to assess the maximum performance achievable in this task when a large dataset is available. Instead, it is necessary to establish whether the considered models produce good representations and to define a hierarchy among such models based on their performance when used in the same scenario and under the same conditions.

Moreover, machine learning models are data-driven: generally, the more samples are used for training, the better performing will be the final model, while also enabling the use of more complex models with a larger number of parameters. Therefore, it is probably more interesting to assess how these representation models behave when used in the more challenging scenario of a smaller training dataset and simple linear classification models.

The news, regardless of the representation model used, are all lowercased. If multiple white space characters (e.g., tabs, new lines, multiple spaces) appear, those are replaced with a single space character.

Two other sub-datasets are obtained from the considered news dataset: one used to classify the news using only the title and another using the first three to five sentences (including the title). Finally, to evaluate the representation models, the dataset is partitioned in a stratified fashion, producing similarly distributed classes between the splits, into a training dataset (70%) and a test dataset (30%): the same train and test splits are used across all tests.

4.1.2 Text Representation Models Considered

We considered the following models for mapping texts into vectors used in the downstream classification model.

1. *Bag-of-words + tf-idf.* The classical bag-of-words model with tf-idf reweighting described in Section 2.1 is taken as a reference point. The implementation used is that provided by the `TfidfVectorizer` class from the scikit-learn library [43]. Language-specific stop words (e.g., “ad”, “in”, “per”) are removed using the NLTK library [44], while corpus-specific stop words are removed by setting the terms’ maximum document frequency to 50% and considering up to $2^{18} \approx 262$k features corresponding to the most frequent words of at least two characters. The resulting vectors are normalized to unitary ℓ_2 norm.

2. *Hashing trick.* The hashing trick detailed in Section 2.1.1 is used as a stateless

approach to the bag-of-words method, which does not store the word-to-index mappings. The implementation is provided by the `HashingVectorizer` class from the scikit-learn library. Similarly to the classical bag-of-words, language-specific stop words are ignored (no corpus-specific stop words are removed), up to 2^{18} features are considered (represented by words of at least two characters), and the resulting vectors are normalized to unitary ℓ_2 norm. In addition, the hash function used is signed, so collisions are likely to cancel out rather than accumulate errors.

3. *Latent semantic analysis.* The document-term matrices produced by the bag-of-words and hashing vectorizers are reduced to a lower dimensional representation of 1000 features using the LSA procedure described in Section 2.1.2. The number of dimensions was arbitrary: [12] suggests values in the low hundreds when LSA is used for similarity searches, but a larger number will probably retain more information while still significantly reducing the dimensionality. The `TruncatedSVD` class from scikit-learn, which employs a randomized algorithm to solve the SVD problem, is used to perform the dimensionality reduction. Similarly to the non-reduced variants, the representations produced by LSA are normalized to the unitary ℓ_2 norm.

4. *Doc2Vec.* The distributed bag-of-words (dbow) Doc2Vec model, described in Section 2.2.2, is trained on the entirety of the Italian Wikipedia [45] using the Gensim library [21]. Given that the setting is similar to [20] apart from using the Italian Wikipedia instead of the English one, the same hyperparameters are used: vector size of 300 units, a window size of 15 words, sub-sample threshold of 10^{-5}, five negative samples and 20 epochs. The improvement to the dbow model provided by Gensim, consisting in jointly learning the word vectors in a skip-gram fashion, is enabled. The learning rate is left at the default value, which starts at $2.5 \cdot 10^{-2}$ and decreases linearly to 10^{-4}. Finally, the inference stage to obtain the vector representation for unseen texts uses the same parameters (i.e., epochs and learning rate) as the training stage.

5. *Sentence-BERT.* A pre-trained $\text{BERT}_{\text{BASE}}$ model [46] is fine-tuned following the Sentence-BERT method detailed in Section 2.3.3, using the SentenceTransformers library [14]. The considered BERT model is pre-trained on an Italian corpus of 81GB comprising the Italian Wikipedia, machine-translated web pages, and the Italian portion of the Common Crawl corpus. The BERT sentence embeddings are obtained by average pooling the last layer's hidden representations. Fine-tuning is performed using the regression objective function (Figure 2.6b) on the STSbenchmark (STSb) dataset translated into Italian [47]. The STSb dataset consists of 8628 sentence pairs split into the train (5749), dev (1500), and test (1379) sets and labeled with a semantic similarity score $\in [0, 5]$. The similarity score has been rescaled to $[0, 1]$

to compute the loss as the MSE between the label and the cosine similarity of the embeddings. Similarly to [14], the model is fine-tuned for four epochs using a batch size of 16, Adam optimizer with a learning rate of $2 \cdot 10^{-5}$ and a linear learning rate warm-up over 10% of the training data. Every 500 optimization steps, the model is evaluated using the dev split, and if the performance improves, the model is saved.

Additionally, for the bag-of-words model and its variants, the performance is evaluated in two different scenarios:

- *In-domain.* The representation model is trained using the training split on which the downstream model is trained. This means that the bag-of-words model's vocabulary contains only the words that appear exclusively in the training split. However, if the hashing trick is employed, no vocabulary is used, and no difference can be observed.

- *Cross-domain.* Similarly to how Doc2Vec[1] and Sentence-BERT were trained, the representation model is learned from an external corpus, and the model is intended to be used as an "off-the-shelf" model to encode texts from different domains. This means that the vocabulary contains a richer set of words from domains that do not necessarily coincide with the domain of the downstream task. Furthermore, LSA finds a reduced space of concepts more general than the one it would obtain from the training set. This may represent a limitation of LSA in the cross-domain scenario: useful concepts for the downstream task may be replaced (in the reduced representation) by concepts orthogonal to the domain of the task at hand, bounding the final classification performance. A random sample of 500k articles from Wikipedia was used to learn the models in this setting.

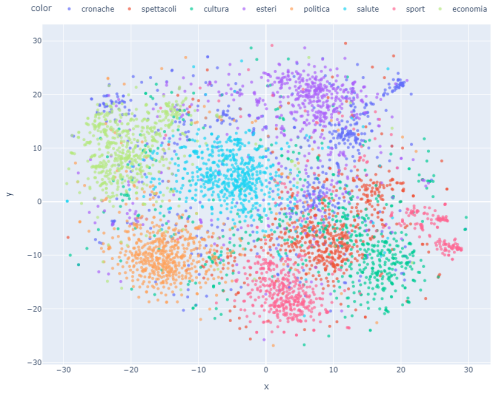
In Figure 4.1, a 2D visualization of the vectorized test news obtained using t-SNE is provided for the reduced hash (in the cross-domain setting), Doc2Vec, and Sentence-BERT models.

4.1.3 Evaluation Methodology

To assess the effectiveness of the text representation models, the vector representation they provide is used as input to a linear model that classifies the news articles according to their topic.

[1]Doc2Vec could have also been analyzed in the in-domain scenario by training the paragraph vectors directly on the samples from the training set. However, in this analysis, we limited ourselves to Doc2Vec in the cross-domain scenario.

(a) Reduced hashing



(b) Doc2Vec

(c) Sentence-BERT

Figure 4.1: 2D visualization of the text representations using different models

For each of the different classes, the performance achieved by the classification model is measured using the class-specific F_1 score defined as the harmonic mean of the precision and recall measures:

$$\text{pre} = \frac{\text{tp}}{\text{tp} + \text{fp}} \qquad \text{rec} = \frac{\text{tp}}{\text{tp} + \text{fn}} \tag{4.1}$$

$$F_1 = \frac{2}{\text{pre}^{-1} + \text{rec}^{-1}} = \frac{\text{tp}}{\text{tp} + \frac{1}{2}(\text{fp} + \text{fn})} \tag{4.2}$$

where tp stands for *true positives* and represents the number of correctly classified samples for the given class, fp stands for *false positives* and represents the number of samples wrongly assigned to the given class, and fn which stands for *false negatives* and represents the number of samples from the given class which have been wrongly assigned to a different class. The Macro-F_1 score, obtained by averaging the F_1 scores of all classes, is taken as an aggregate measure of the performance achieved by the classifier.

The `SGDClassifier` class from scikit-learn is used. `SGDClassifier` implements

multiple regularized linear classifiers optimized via SGD training. The linear model used and other hyperparameters are selected by performing an exhaustive search over a grid of values, and the performance is evaluated in a cross-validated fashion: the training data is split into $k = 5$ folds, and the model is trained k times leaving out each time a different fold and finally the performance for a hyperparameters configuration is obtained by averaging the performance obtained in each of the k left out folds. Grid search with cross-validation functionality is provided by the `GridSearchCV` class from scikit-learn.

More in detail, the following hyperparameters are optimized: loss $\in$ {hinge, cross-entropy}[2], regularization penalty $\in \{\ell_1, \ell_2, \text{elastic-net}\}$ with the latter being a linear combination of the first two, the regularization constant $\alpha = 10^{-i}$ for $i \in 1, \ldots, 5$ which also controls the learning rate lr at step t according to $lr = 1/[\alpha(t + t_0)]$ and finally the number of epochs without an improvement in performance in a validation split (obtained from the training data) before stopping training $\in$ {5, 10, 20, 30}.

4.2 Text Representations Inversion Attack

This section describes the inversion attack implemented and mounted against the text representations produced by some of the models analyzed in Section 4.1.2. By only accessing the vectorized text representation, the inversion attack aims to recover the words appearing in the plaintext. The attack model is designed to be as universal as possible to be easily adapted to any text representation with minimal to no modifications. However, this design principle probably comes at the cost of limiting the attack's maximum achievable performance: a well-crafted attack that targets a particular representation model or accesses more information with respect to the analyzed attack could potentially achieve higher performance and hence represent a greater privacy threat. Therefore, the inversion results obtained using the proposed attack should be taken as a *lower bound* for the information that can be recovered from a vectorized text representation produced by the considered models.

Section 4.2.1 describes the setting that motivated the analysis, and Section 4.2.2 defines the information available to the attacker while introducing the notation used. Then, Section 4.2.3 describes the architecture and functioning of the inversion attack model.

[2]Hinge loss results in a linear SVM, while cross-entropy in a logistic regression model.

4.2.1 Motivation and Setup

The scenario of interest that motivated the study is the following. A customer, called A, turns to the company C that provides ML-based NLP services. The ML models are trained on the customer's private data. However, A — in an attempt to achieve confidentiality of its data — does not want to send its documents as plaintexts but prefers to produce the vectorized representations of such documents and send them to C. For this purpose, C provides its customers with a text vectorization model $\Psi(\cdot)$.

The goal of A is to maintain the confidentiality of its data and obtain a quality model (i.e., with good performance) from C. On the other hand, C aims to satisfy the client by producing a quality downstream model from the client's vectorized representations computed using a simple and fast vectorization model $\Psi(\cdot)$, assuming that the performance delta compared to more complex vectorization models is negligible.

In the described setup, the role of the attacker can be assumed by different entities:

1. C itself: from A's point of view, the *possibility* that C may be able to recover any information from the vectorized representations may represent a potential risk, even if C itself is not interested in performing any attack.

2. An external entity to C (e.g., another customer) called B. In this case, B does not have direct access to the vectorized representations if a security measure is in place to avoid their leak; therefore, no attack should be possible. However, exclusively relying on other security measures may represent a risk considering that the security provided by a complex system is only as strong as the security provided by its weakest component, which is usually a human. Assume, for example, that C ensures A that A's private data are under no circumstances revealed to B. For this purpose, C has a security mechanism to prevent this leakage. However, the employee of C who manages security decides to leak the vectorized representations. B can now carry out the attack even if C uses a cutting-edge security mechanism. On a side note, a vectorized representation that prevents as much information leakage as possible would act as an additional line of defense in the case of data leakage.

Furthermore, assuming that C provides the same downstream model to multiple clients, a secure representation of texts could allow C to jointly train a model using data from various clients. However, this scenario opens up new challenges, such as potential training data memorization, which can make the model susceptible to

training data extraction (Section 3.2) or membership inference attacks (Section 3.3). Therefore, this scenario should be adequately addressed, for example, using differentially private mechanisms (Section 3.5) in a federated learning setup [48].

4.2.2 Threat Model

In the scenario of interest, the attacker, other than the text representations they wish to invert $\mathcal{D}_{\text{target}} = \{(\Psi(x_i^*))\}_i$, where x_i^* is the unknown plaintext and $\Psi(\cdot)$ is the model producing the representations, is only required to have access to a dataset of plaintext to vectorized representation mappings $\mathcal{D}_{\text{train}} = \{(x_i, \Psi(x_i))\}_i$. The attacker then uses $\mathcal{D}_{\text{train}}$ to train the attack model $\mathcal{A}$, as will be described in Section 4.2.3.

$\mathcal{D}_{\text{train}}$ could be obtained by the attacker in different ways. For example, the attacker could build $\mathcal{D}_{\text{train}}$ if they happen to gain query access to the model $\Psi(\cdot)$ computing the representations: this may happen either if the attacker is C since it is the one providing $\Psi(\cdot)$, or B if $\Psi(\cdot)$ is (reasonably) the same among all customers or it is a publicly available model. Another scenario, although unlikely, is that $\mathcal{D}_{\text{train}}$ is somehow leaked by C or another customer.

Furthermore, $\mathcal{D}_{\text{train}}$ could come from the same distribution ($\mathcal{D}_{\text{train}}^{\text{in}}$) as $\mathcal{D}_{\text{target}}$ or from a different distribution ($\mathcal{D}_{\text{train}}^{\text{cross}}$). For example, assuming that the attacker wants to invert vectorized emails using $\Psi(\cdot)$ and exchanged by A's employees, the former case would occur if the attacker has access to a dataset of emails from A, while the latter case would occur if the dataset is obtained from a different domain (e.g., data scraped from the Web).

The inversion attack $\mathcal{A}$ is formulated as a learning problem that is consequently implemented by means of an ML model. Given the vectorized representation $\Psi(x^*)$ of an unknown plaintext x^*, $\mathcal{A}$ is trained to predict the *unordered* set of words $\mathcal{W}(x^*)$ appearing in x^* from the attacker's vocabulary $\mathcal{V}_{\text{attack}}$. Namely, given $\Psi(x^*) \in \mathcal{D}_{\text{target}}$:

$$\mathcal{A} : \Psi(x^*) \mapsto \mathcal{W}(x^*) \subseteq \mathcal{V}_{\text{attack}} \tag{4.3}$$

The attacker vocabulary $\mathcal{V}_{\text{attack}}$ may, in principle, be any collection of words. However, recovering uninteresting language-specific stop words may not be of interest to the attacker. At the same time, recovering rare words can become challenging and possibly not feasible. However, it should be noted that even a combination of common words, if linked to a particular individual or organization (e.g., words that reveal the business strategy of a company), may constitute a privacy breach.

The attacker need not know exactly what $\Psi(\cdot)$ is nor be able to access its parameters. With this limited information available to the attacker, $\mathcal{A}$ falls into the

black-box attack category. As previously observed, this attack was chosen to be easily applicable to any text vectorization technique. An attack that accesses the parameters of $\Psi(\cdot)$ in a *white-box* fashion or which exploits some additional information (e.g., the exact nature of $\Psi(\cdot)$) could theoretically be more performing and hence represent a greater threat to privacy.

4.2.3 Attack Model

The text representations inversion attack $\mathcal{A}$ introduced in Section 4.2.2 is implemented as a slight variation of the black-box sentence embedding inversion attack proposed in [9][3].

Differently from [9]: (1) the attack is carried out against multiple text representation models not limited to deep learning embedding models; (2) $\mathcal{A}$ has no access to the vocabulary used by the representation model $\Psi(\cdot)$; (3) $\mathcal{A}$ is applied to the representation of longer texts (i.e., paragraphs) and not just sentences; (4) Uninteresting stop words (and punctuation) and are considered in $\mathcal{V}_{\text{attack}}$ and therefore not returned by $\mathcal{A}$.

As mentioned previously, $\mathcal{A}$ is implemented as an ML model that, given the vectorized representation $\Psi(x)$, learns to predict the *unordered* set of words $\mathcal{W}(x) \subseteq \mathcal{V}_{\text{attack}}$ appearing in the plaintext x. As discussed in [9] this task can be approached in two ways: as a *multi-label classification* problem where the words appearing in x are predicted simultaneously and independently from one another, or as a *set prediction* problem in which words in x are predicted sequentially while conditioning at each step on the set of previously predicted words. The latter approach was found to yield much better performance (similar to the white-box attack) and is consequently implemented by $\mathcal{A}$.

To tackle the inversion attack as a set prediction problem, an RNN is trained to predict the next word in the set $\mathcal{W}(x)$ by conditioning on the text vectorization $\Psi(x)$ and the set of currently predicted words. At each step $t = 1, \ldots, L$ the network outputs a distribution over the words in $\mathcal{V}_{\text{attack}}$ and the word at step t is greedy sampled from such distribution (i.e., the word with the highest probability is selected). The final set of recovered words $\mathcal{W}_{\text{rec}}(x)$ is obtained as the union of the words predicted at each of the L steps.

Given a single $(\Psi(x), \mathcal{W}(x))$ training mapping, the network with parameters Θ

[3]The white-box sentence embedding inversion attack proposed in [9] was described in Section 3.4.1.

is optimized by minimizing the following sample loss:

$$\mathcal{L}(\Theta) = \sum_{t=1}^{L} \frac{1}{|\mathcal{W}_t|} \sum_{w \in \mathcal{W}_t} -\log P(w|\Psi(x), \mathcal{W}_{<t}; \Theta) \tag{4.4}$$

where $\mathcal{W}_t$ is the set of words in the plaintext x *left to predict* at step t, $\mathcal{W}_{<t}$ is the set of words in x *predicted* up to step $t-1$. Note that $\mathcal{W}_0 = \mathcal{W}(x)$, $\mathcal{W}_{<1} = \emptyset$, and $\mathcal{W}_{\text{rec}}(x) = \mathcal{W}_{<L+1}$.

As can be easily observed from Equation 4.4, the sample loss for a training mapping $(\Psi(x), \mathcal{W}(x))$ results from the summation of the average negative log-likelihoods of the words left to predict at each prediction step. Therefore, by minimizing the loss function, the attack model is trained to assign high likelihoods (and, accordingly, to sample) the words that appear in $\mathcal{W}(x)$.

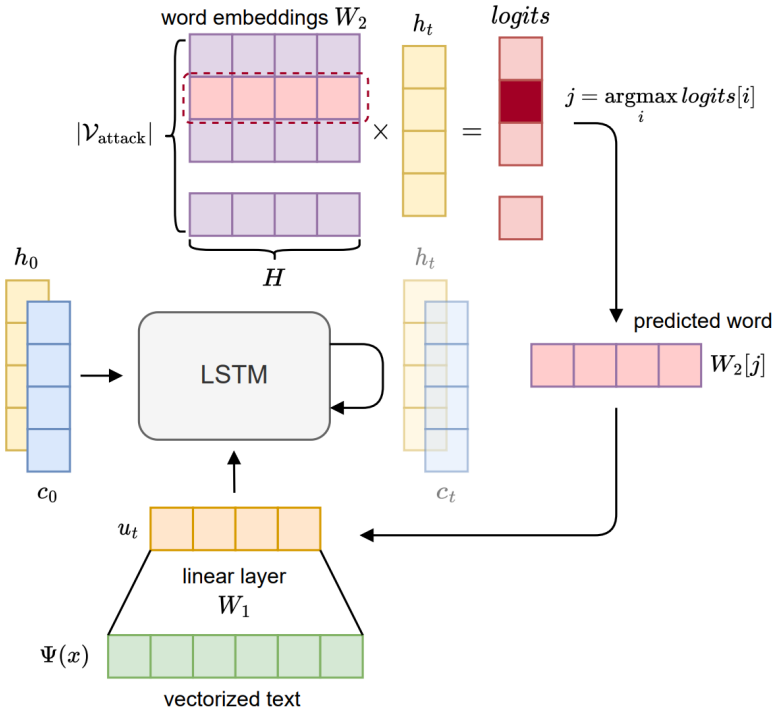


Figure 4.2: Architecture of the text representations inversion attack model. First, the vectorized representation $\Psi(x)$ of a text x is projected into a compressed representation u_1. Then, the model uses an RNN implemented with an LSTM cell to predict sequentially the words appearing in x. In particular, the first input to the RNN is u_1, while the subsequent input is the embedding learned by the model and corresponding to the word predicted at the previous step.

The learned embedding of each predicted word is fed back into the LSTM to perform the consequent prediction step.

The architecture of the attack model $\mathcal{A}$ — consisting of a linear layer with weight

matrix W_1, an RNN implemented using an LSTM cell, and a matrix W_2 of word embeddings — is depicted in Figure 4.2. The parameters Θ learned during training are the linear layer weights W_1, the LSTM cell weight matrices and initial states (c_0, h_0), and the entire matrix of word embeddings W_2.

The linear layer is used to project the vectorized text representation of arbitrary length $\Psi(x)$ into a network-specific representation u_1 of H units, which corresponds to the initial input of the LSTM cell. The idea is to map the representations produced by $\Psi(\cdot)$, which may have an arbitrary length (even extremely large) and different normalization techniques applied (e.g., ℓ_1 or ℓ_2), into a vector space that makes it comparable to the internal states and word embeddings used by the network. No activation function is used on the output of the linear layer.

The LSTM cell is a computing unit that controls the information that propagates over time in the RNN by selectively adding or removing information to the internal state represented by the cell state and hidden state vectors. LSTMs were introduced to better deal with the vanishing gradient problem common in RNNs, which makes it difficult to deal with long-term sequential dependencies.

At each time step t, the LSTM cell takes in input the current input u_t, the previous step cell state c_{t-1} and the hidden state h_{t-1}, and outputs the new internal state c_t, h_t. While the cell state c_t and the hidden state h_t are required to have the same dimension, the input u_t does not need to have the same number of dimensions. For ease of development, as will become clear later, in our implementation, the internal states c_t and h_t, and the input u_t share the same number of dimensions H.

The word embedding matrix is a $|\mathcal{V}_{\text{attack}}| \times H$ matrix that stores for each word in the vocabulary $\mathcal{V}_{\text{attack}}$ the corresponding embedding learned by the model. On a side note, this matrix can be simply viewed as a weight matrix of a linear classification layer with H input features and $|\mathcal{V}_{\text{attack}}|$ possible output classes. The name is used to highlight the difference with the previous linear layer and also to suggest a possible improvement not analyzed in this work: instead of starting from a random initialization of Θ, a better initialization of the weights to speed up training and possibly improve performance could consist of using pre-trained word embeddings (e.g., Word2Vec) as rows in the word embedding matrix W_2 and accordingly change the value of H.

Algorithm 1 reports the pseudocode with the steps required to perform a forward pass through the network and correspondingly compute the loss according to Equation 4.4 for a single training mapping $(\Psi(x), \mathcal{W}(x))$. Note that the training procedure uses the same pseudocode, with the difference of considering an appropriately sized batch instead of a single mapping. On the other hand, the inference procedure to recover the words $\mathcal{W}_{\text{rec}}(x^*)$ from the vectorization $\Psi(x^*)$ of an unknown

plaintext x^* uses the same steps apart from the loss computation, which is omitted.

Algorithm 1: Forward pass with loss computation through the text representations inversion attack model.

Input : vectorized text $\Psi(x)$, set of words in the plaintext $\mathcal{W}(x) \subseteq \mathcal{V}_{\text{attack}}$
Output: recovered words $\mathcal{W}_{\text{rec}}(x)$ from $\Psi(x)$, sample loss $\mathcal{L}$

1 $\mathcal{L} \leftarrow 0$, $\mathcal{W}_0 \leftarrow \mathcal{W}(x)$, $\mathcal{W}_{<1} \leftarrow \emptyset$ `// loss, words left, words predicted`
2 Compute initial input $u_1 \leftarrow W_1\Psi(x)$
3 $done \leftarrow False$ `// Set to True when [EOS] token is predicted`
4 $t \leftarrow 1$
5 **while not** $done$ **and** $t \leq L$ **do**
6 Compute internal states $c_t, h_t \leftarrow \text{LSTM}(u_t, c_{t-1}, h_{t-1})$
7 Compute similarities $logits \leftarrow W_2 h_t$
8 Predict a word $j = \text{argmax}_i\, logits[i]$
9 Set next input $u_{t+1} \leftarrow W_2[j]$ `// Select j-th row of matrix` W_2
10 $\mathcal{W}_t \leftarrow \mathcal{W}_{t-1} \setminus \{j\}$
11 $\mathcal{W}_{<t+1} \leftarrow \mathcal{W}_{<t} \cup \{j\}$
12 Compute probabilities $probs \leftarrow \text{softmax}(logits)$
13 $\mathcal{L} \leftarrow \mathcal{L} - \frac{1}{|\mathcal{W}_t|} \sum_{i \in \mathcal{W}_t} \log probs[i]$
14 **if** j *is the index of* `[EOS]` **then**
15 $done \leftarrow True$
16 $t \leftarrow t + 1$
17 $\mathcal{W}_{\text{rec}}(x) \leftarrow \mathcal{W}_{<L+1}$
18 **return** $\mathcal{W}_{\text{rec}}(x)$, $\mathcal{L}$

Although the steps in Algorithm 1 are quite straightforward, it is worth making a few observations.

In order to predict the word at step t, the word embedding matrix W_2 is multiplied to the right by the hidden representation h_t produced by the LSTM cell (line 7). This matrix multiplication corresponds to computing the dot product between the rows of W_2, which are word embeddings, and h_t. The dot product between embeddings can be used as a measure of similarity between the objects they represent[4]. Therefore, each position $i \in [0, |\mathcal{V}_{\text{attack}}| - 1]$ of the resulting column vector contains a value measuring the similarity between the word in $\mathcal{V}_{\text{attack}}$ corresponding to the index i and the hidden state h_t. The resulting vector, called *logits*[5] since it is a vector of non-normalized predictions (our similarity values), is used to sample the word at the given step (line 8).

[4]On a side note, the dot product of two vectorized text representations with unitary ℓ_2 norm is equal to their cosine similarity as defined in Equation 2.6.

[5]In the ML field, *logits* generally refers to a vector of non-normalized predictions produced by a classification model, which is then passed to a normalization function. For example, if the model solves a classification problem, the logits vector is typically normalized to a vector of probabilities, with one value per class, using the softmax function [49]

Once the word is predicted, the corresponding word embedding is selected by extracting the associated row from the matrix W_2 (line 9). This embedding is then used as the next LSTM input u_{t+1}. Now it should be clear why both the internal states (c_t and h_t), the LSTM input u_t, and the internal word embeddings all share the same number of units H. These observations motivated the distinction introduced earlier between W_1 and W_2.

In order to compute the loss according to Equation 4.4, at each prediction step t the probability distribution over the words in $\mathcal{V}_{\text{attack}}$ is computed using the softmax normalization function (line 12). More in detail, the probability $P(w|\Psi(x), \mathcal{W}_{<t}; \Theta)$ for the word $w \in \mathcal{V}_{\text{attack}}$ assigned to the index i is computed as follows:

$$probs[i] = \frac{\exp(logits[i])}{\sum_{j=0}^{|\mathcal{V}_{\text{attack}}|-1} \exp(logits[j])} \tag{4.5}$$

As a final note, not mentioned earlier, notice that the prediction stops, and consequently, the loss does not decrease as soon as the model predicts the end-of-sequence token `[EOS]`.

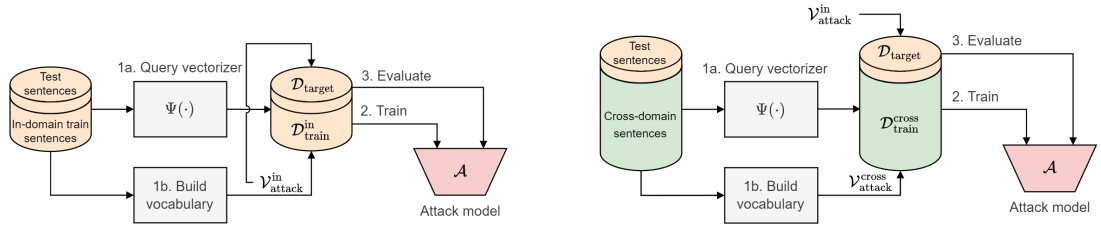
4.3 Inversion Attack Implementation

This section describes the procedure followed to train and evaluate the inversion model $\mathcal{A}$. In particular, Figure 4.3 sketches the various steps followed when $\mathcal{A}$ is trained on a dataset $\mathcal{D}^{\text{in}}_{\text{train}}$ from the same domain of the target text vectorizations $\mathcal{D}_{\text{target}}$ used to evaluate the inversion performance achieved by $\mathcal{A}$ (Figure 4.3a) and when $\mathcal{A}$ is trained on a dataset $\mathcal{D}^{\text{cross}}_{\text{train}}$ from a different domain compared to $\mathcal{D}_{\text{target}}$ (Figure 4.3b).

Section 4.3.1 describes the procedure followed to generate the in-domain, cross-domain, and test datasets; Section 4.3.2 describes the evaluation metrics used to assess the performance achieved by $\mathcal{A}$; and finally, Section 4.3.3 details the hyper-parameter tuning and training procedures followed to optimize the parameters Θ of $\mathcal{A}$.

4.3.1 Datasets Generation

As mentioned in Section 4.2.2, the in-domain analysis setup exemplifies the scenario in which the attacker managed to access a dataset of plaintexts and their vectorizations computed by $\Psi(\cdot)$ and aim to invert the vectorized representation of some other *unknown* plaintexts from the *same* distribution of the known plaintexts. Therefore, known and unknown plaintexts will have a similar structure, vocabulary, and topics



(a) In-domain scenario.

(b) Cross-domain scenario.

Figure 4.3: Training and evaluation pipeline for the text representations inversion attack model. Figure a illustrates the pipeline using in-domain training data while Figure b using cross-domain training data. Notice that the attack vocabularies are different in the two scenarios because they are generated from different data. However, the test sets are still the same.

covered. On the other hand, the cross-domain analysis setup represents the more likely scenario in which the attacker cannot access similar plaintexts to those they wish to recover but instead use a dataset from a different distribution for training the inversion model $\mathcal{A}$. An example of in-domain data could be emails exchanged by the employees of a company's business unit. In contrast, the corresponding cross-domain data could be texts scraped from the Web.

Since we would like to compare how the inversion model performs in the two scenarios (Section 5.2.1 and Section 5.2.2), the same test set is used to evaluate the performance in both cases, as can be observed in Figure 4.3. Furthermore, in this analysis, we tried to replicate the two scenarios exclusively using publicly available data from the Web. In particular, in the experiments, the in-domain, cross-domain, and test datasets are generated from a collection of news articles from the same Italian newspaper used in Section 4.1. The procedure followed to generate $\mathcal{D}^{\text{in}}_{\text{train}}$, $\mathcal{D}^{\text{cross}}_{\text{train}}$, and $\mathcal{D}_{\text{target}}$, depicted in Figure 4.4 and partially (steps 1a and 1b) in Figure 4.3, is now described.

Each news article labeled with class C was selected to isolate a particular domain from the newspaper dataset. Articles labeleld with class C were used to generate the in-domain train data and the test data used to evaluate performance in both the in-domain and cross-domain scenarios; any other news article with a label different from C was used to generate the cross-domain train data (step 1 of Figure 4.4). We chose C = Economy because it provided a sufficient number of articles (21179) to generate enough samples to effectively train $\mathcal{A}$ and evaluate its performance in the two scenarios while distinguishing itself sufficiently from all other classes: Politics (20410), Sport (18988), Entertainment (11679), Arts (8630), Health (5881), School (930), and Science (89). News articles without a specific domain, namely those

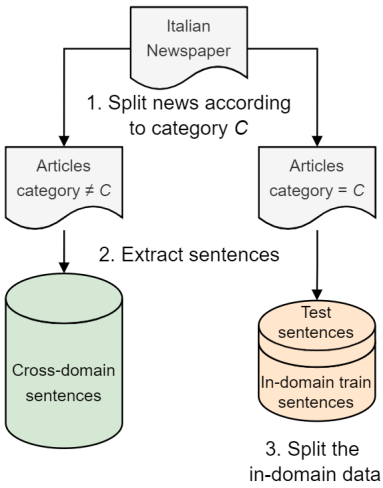


Figure 4.4: In-domain and cross-domain datasets generation strategy. Newspaper articles with category C = Economy are selected to generate in-domain data. All the other articles (e.g., Entertainment, Sport, Health) make up the cross-domain train data. The articles are then tokenized in sentences, which are grouped into paragraphs. The in-domain paragraphs are split into in-domain train data and test data.

labeled with class General (32500) and World (29645), were not considered in the cross-domain dataset, as they are too generic and tend to overlap excessively with C[6].

One may argue that using the same underlying dataset to produce both in-domain and cross-domain data may yield two datasets that are still too similar to each other. However, it is also true that in the cross-domain scenario, if the attacker has access to $\Psi(\cdot)$, they could generate an arbitrary large dataset (by scraping the Web), which may happen to include samples with a domain similar to the target plaintexts. Additionally, suppose the attacker knows the nature of the underlying target plaintexts. In that case, they could carefully produce their cross-domain train dataset by scraping or artificially generating texts that match as much as possible the underlying domain. However, the dataset generation was performed in such a way, mainly for simplicity and lack of better alternatives: no dataset analogous to the Enron email dataset (which could have been used as in-domain data) exists for the Italian language.

The texts of the news articles are tokenized into sentences using the NLTK

[6]On a side note, from Figure 4.1, which reports the 2D visualization of the vectorized representation for a subset of articles (600 randomly drawn samples per class not including School and Science), it can be observed how General and World effectively overlap with the others, while all other classes are mostly represented by well-separated clusters.

library [44] (step 2 of Figure 4.4). The sentences are then lowercased, and if multiple white space characters (e.g., tabs, new lines, multiple spaces) appear, those are substituted with a single space character. Only those sentences with a number of words (of at least two characters) ranging between 8 and 40 are kept in the sentence dataset. The sentences from the same article, satisfying the length requirements, are aggregated into artificial paragraphs of at least 128 words and up to 168 words (minimum length of a paragraph plus the maximum length of a sentence): each paragraph is in a one-to-many relationship with its sentences. This was done to investigate how encoding longer pieces of text instead of individual sentences affects the performance of the inversion attack (Section 5.2.2).

The in-domain paragraphs dataset (i.e., generated from C articles) is split into a train dataset (166377 sentences) and a test dataset (71398 sentences) (step 3 of Figure 4.4). Note that, despite the partitioning being performed at the paragraph level, in the in-domain scenario, some sentences from a given news article may go into the training dataset. In contrast, the remaining sentences from other paragraphs of the same article may go into the test dataset. All sentences from news articles with a category different from C (849692 sentences) constitute the cross-domain train dataset. How performance scales with the number of in-domain training samples available to the attacker is analyzed in Section 5.2.4.

Once the in-domain train, the cross-domain train and the test datasets are generated, $\mathcal{D}_{\text{train}}^{\text{in(cross)}}$ and $\mathcal{D}_{\text{target}}$ can be constructed. In particular, the train and target datasets for the cross-domain scenario are defined as

$$\mathcal{D}_{\text{train}}^{\text{cross}} = \left\{ (\Psi(x_i), \mathcal{W}(x_i)) \;\middle|\; \begin{array}{l} x_i \text{ train cross-domain plaintext} \\ \mathcal{W}(x_i) \subseteq \mathcal{V}_{\text{attack}}^{\text{cross}} \end{array} \right\}$$

$$\mathcal{D}_{\text{target}} = \left\{ (\Psi(x_i), \mathcal{W}(x_i)) \;\middle|\; \begin{array}{l} x_i \text{ test plaintext} \\ \mathcal{W}(x_i) \subseteq \mathcal{V}_{\text{attack}}^{\text{in}} \end{array} \right\}$$

and the corresponding in-domain variants are analogous (see Figure 4.3). More in detail, such datasets are generated by:

(a) Computing the vectorized representations by querying the model $\Psi(\cdot)$ (step 1a of Figure 4.3). In the experiments, the following six vectorization models, used in Section 4.1.2, are considered: cross-domain bag-of-words with tf-idf reweighting and hashing vectorizer with their LSA reduced versions, Doc2Vec and Sentence-BERT.

(b) Building the vocabulary $\mathcal{V}_{\text{attack}}$ considering from the training sentences the 20k most frequent words of at least two characters not in the NLTK list of language-specific stop words (step 1b of Figure 4.3). Note that since in the two scenarios

the training datasets are different, the corresponding vocabularies $\mathcal{V}^{\text{in}}_{\text{attack}}$ and $\mathcal{V}^{\text{cross}}_{\text{attack}}$ will not be the same. However, when comparing performance in the two scenarios (other than the same test sentences), it makes sense to use $\mathcal{V}^{\text{in}}_{\text{attack}}$ in both cases, as it may contain sensitive words that do not appear in the cross-domain training dataset and therefore in $\mathcal{V}^{\text{cross}}_{\text{attack}}$.

4.3.2 Inversion Performance Evaluation

The inversion model $\mathcal{A}$ is evaluated on $\mathcal{D}_{\text{target}}$. Performance is measured in terms of precision (percentage of correctly recovered words), recall (percentage of recovered words), and their harmonic mean expressed by the F_1 score (Equation 4.2).

Adapting the precision and recall definitions from Equation 4.1 — where *true positives* indicate correctly recovered words from $\Psi(x)$, *false positives* indicate recovered words which do not appear in x, and *false negatives* indicate words in x not recovered from $\Psi(x)$ — it results:

$$\text{pre} = \frac{|\mathcal{W}_{\text{c}}(x)|}{|\mathcal{W}_{\text{rec}}(x)|} \qquad \text{rec} = \frac{|\mathcal{W}_{\text{c}}(x)|}{|\mathcal{W}(x)|} \tag{4.6}$$

where $\mathcal{W}_{\text{c}}(x) = \mathcal{W}_{\text{rec}}(x) \cap \mathcal{W}(x)$ is the set of correctly recovered words. As an aggregate measure of performance, the mean values of pre, rec, and F_1 across all samples in $\mathcal{D}_{\text{target}}$ are considered.

The above performance metrics treat all words equally. However, recovering words that appear with high frequency and carry less information is a less challenging task (and alarming from a security perspective) than recovering words that appear with a lower frequency (e.g., proper names). For this reason, as additional performance metrics, a generalization of the precision and recall metrics of Equation 4.6 are considered. More in detail, each word $w \in \mathcal{V}_{\text{attack}}$ is assigned a weight α_w inversely proportional to its frequency in $\mathcal{D}_{\text{target}}$, and precision and recall are obtained as:

$$\text{pre}_w = \frac{\sum_{w \in \mathcal{W}_{\text{c}}(x)} \alpha_w}{\sum_{w \in \mathcal{W}_{\text{rec}}(x)} \alpha_w} \qquad \text{rec}_w = \frac{\sum_{w \in \mathcal{W}_{\text{c}}(x)} \alpha_w}{\sum_{w \in \mathcal{W}(x)} \alpha_w} \tag{4.7}$$

Correspondingly, F_1^w is obtained by computing the F_1 score of pre_w and rec_w. Observe that if $\alpha_w = 1 \; \forall \; w \in \mathcal{V}_{\text{attack}}$, pre_w and rec_w correspond to pre and rec.

The weights are computed similarly to the idf factors (Equation 2.1) in the tf-idf reweighting of the bag-of-words representation model (Section 2.1):

$$\alpha_w = \log \frac{|\mathcal{D}_{\text{target}}| + 1}{\text{c}(w) + 1} \tag{4.8}$$

where $\text{c}(w)$ is the number of plaintexts in $\mathcal{D}_{\text{target}}$ that contain w. The $+1$ at the

numerator and the denominator represents an additional (fictitious) sample in $\mathcal{D}_{\text{target}}$ that contains all the words in $\mathcal{V}_{\text{attack}}$ and is done to avoid divisions by zero, since some words in $\mathcal{V}_{\text{attack}}$ could potentially never appear in $\mathcal{D}_{\text{target}}$.

4.3.3 Inversion Model Training

The inversion model $\mathcal{A}$ was implemented using the PyTorch framework [50]. Amazon SageMaker, a fully managed ML service that provides the tools and infrastructure to train and deploy ML models [51], was used to optimize the hyperparameters and train $\mathcal{A}$. In particular, the high-level interfaces offered by the SageMaker Python SDK library [52] were used to work with the PyTorch framework using the SageMaker tools. Each training job was run on an ml.g4dn.xlarge SageMaker instance powered by an Nvidia Tesla T4 GPU.

Regardless of the model $\Psi(\cdot)$ that produces the vectorizations, the same number H of internal units of $\mathcal{A}$ was used for all models. Apart from treating all vectorization models equally, the rationale behind this decision is that, even though the vectorized representation dimensionality dramatically differs from one model to another, all the models encode a similar amount of information contained in the plaintext. How the model encodes such information, either by simply counting the words appearing in the plaintext or by obtaining an abstract representation after many processing layers, should not greatly vary the information content carried by the final representation.

The inversion model $\mathcal{A}$ was trained on $\mathcal{D}_{\text{train}}$. The number of prediction steps L was determined by computing the mean value of words to recover $|\mathcal{W}(x)|$ over each sample in $\mathcal{D}_{\text{train}}$.

The training procedure iterates over the training data, using batches of B samples for a maximum of 100 epochs[7]. Training stops if no loss improvement is observed after a patience of 10 epochs on a validation split (10%) set aside from $\mathcal{D}_{\text{train}}$. The model parameters Θ are optimized using Adam with learning rate γ. The parameters Θ are saved whenever $\mathcal{A}$ achieves a new performance maximum, as measured by the average F_1 score over all samples in the same validation split used for early stopping.

Weight decay and dropout are used to regularize training and avoid overfitting the training data. Weight decay is used to penalize, proportionally to the factor λ, large values of $\|\Theta\|_2$. On the other hand, dropout, which consists of randomly dropping some units to avoid a portion of them becoming much more influential than others, is applied before every LSTM input u_t with probability p.

[7]Even though the maximum number of epochs was set to 100, the execution time for each training job was limited to a maximum of 12 hours. This external limitation had an effect only when the inversion model was trained against some of the representation models in the cross-domain setting due to a large amount of training data available.

The hyperparameters for $\mathcal{A}$ (H) and for the training procedure (B, γ, λ, p) were optimized using a Bayesian search.

As it explores possible hyperparameter configurations, Bayesian search keeps track of previous evaluation outcomes returned by an objective function that measures the quality of the ML model when trained on a particular configuration. Such outcomes are used to build a probabilistic model that maps possible hyperparameter configurations to the score assigned by the objective function (regression-like problem). The Bayesian search then uses the probabilistic function to make informed guesses about which parameter combinations are likely to maximize the objective function and yield quality ML models.

The efficient way Bayesian search explores the hyperparameter landscape should (theoretically) enable it to find: (1) good configurations in fewer trials with respect to naively sampling configurations via a random search and (2) better configurations if the hyperparameter space is too large for an exhaustive grid search, requiring one to limit the search over a smaller space.

Hyperparameter optimization was performed only on the inversion model $\mathcal{A}$ trained to invert the embeddings produced by Sentence-BERT in the in-domain scenario. The choice was arbitrary, but the reasoning is that it represents the most *challenging* setup: text representations produced by a deep model, while the attacker has a limited amount of data to train $\mathcal{A}$, possibly leading to overfitting. The objective function used to evaluate the hyperparameter configuration was the mean F_1 score over all samples in a validation split set aside (20%) from the training data $\mathcal{D}^{\text{in}}_{\text{train}}$.

Despite the training procedure running for a maximum of 100 epochs (with early stopping), during hyperparameter tuning, each training job was run for a maximum of 30 epochs, eventually early stopped by the Bayesian optimizer. This choice was made assuming that given an upper bound to the computational time (and cost), it is probably better to test more hyperparameter configurations for fewer epochs than to test fewer configurations for more epochs. Additionally, if a hyperparameter configuration is good at the start of training, it should still be, even if training continues over the initial 30 epochs.

More in detail, the hyperparameters were tuned in the following ranges: number of internal units $H \in \{384, 512, 768\}$, batch size $B \in \{64, 128, 256\}$, learning rate $\gamma \in [10^{-4}, 10^{-2}]$, weight decay factor $\lambda \in [10^{-5}, 10^{-2}]$, probability of dropout $p \in \{10\%, 20\%, 30\%\}$.

After 32 trials, the optimal parameters found and, consequently, used in all the following training jobs were $H = 512$, $B = 64$, $\gamma = 4 \cdot 10^{-4}$, $\lambda = 10^{-4}$, $p = 20\%$.

Chapter 5

Results

This chapter covers the results of the empirical analyses performed. In particular, Section 5.1 reports the results of the classification benchmark described in Section 4.1, whereas Section 5.2 reports the inversion results on the vectorized text representations obtained using the attack model described in Section 4.2 and implemented as outlined in Section 4.3.

5.1 Classification Benchmark Results

The classification performance obtained for the various representation models described in Section 4.1.2, using the news articles datasets described in Section 4.1.1 and the methodology detailed in Section 4.1.3, is reported in Table 5.1. In particular, performance is measured using the Macro-F_1 score, and, when considered, the difference in performance between the *in-domain* and *cross-domain* scenarios is reported.

Unsurprisingly, the deep model Sentence-BERT outperforms all the other models considered in the cross-domain scenario. However, the improvement is evident only when the news titles are used. At the same time, it is much smaller when the first few sentences or the entire text (eventually truncated to the maximum number of WordPiece tokens accepted by BERT) are used to perform the classification. Quantitatively, the relative improvement over the reference bag-of-words + tf-idf with news titles is 7.13% and decreases to 0.51% and 0.42% with the first sentences or the entire text, respectively.

Although Doc2Vec has the edge over the reference model when used with titles, the latter outperforms Doc2Vec with the other two datasets. However, it should be noted that, as described in Section 4.1.2, the hyperparameters of the Doc2Vec inference stage were not optimized; therefore, the performance with Doc2Vec could

Model	In-domain			Cross-domain		
	Titles	Sentences	Text	Titles	Sentences	Text
Bag-of-words + tf-idf	72.13	82.27	85.23	72.59	82.35	84.67
Hashing	71.22	81.08	84.21	71.22	81.08	84.21
Reduced bag-of-words	68.24	81.36	83.90	63.62	77.63	82.71
Reduced hashing	68.33	80.81	84.31	62.23	77.19	82.24
Doc2Vec	-	-	-	74.64	81.43	83.13
Sentence-BERT	-	-	-	77.77	82.77	85.03

Table 5.1: Text classification benchmark results. Performance is measured in terms of Macro-F_1 score when classifying news articles according to the title, the first sentences, and the entire text. In the in-domain setting, the vectorizer is learned from the training split the classifier is trained on. In the cross-domain setting, the vectorizer is learned from an external corpus.

be potentially improved.

Although the hashing vectorizer, compared to bag-of-words, is prone to collision errors, the performances achieved by the former are not too far off from the latter. Quantitatively, the relative decrease in performance is -1.89% with titles, -1.54% with the first sentences, and -0.54% with the entire text.

The reduced representations obtained with LSA significantly decrease the performance achieved by the classification model. Especially in the cross-domain scenario, this reduction in performance is quite evident when used with titles and the first few sentences, suggesting that LSA is probably not well suited in such a setting of short texts with cross-domain representation.

In two out of three datasets, the reference bag-of-words model in the cross-domain scenario outperforms the same model in the in-domain scenario. Moreover, in this classification scenario, the simple bag-of-words model managed to remain competitive when compared to a deep representation model (except when used on very short texts, i.e., titles), demonstrating that when used in a classification setting, even though it is very simple, it can still manage to achieve good performance.

5.2 Inversion Results

This section reports the inversion results on the vectorized text representations obtained using the attack model $\mathcal{A}$ described in Section 4.2 and following the methodology outlined in Section 4.3. It is important to note that, given the nature of the attack considered, the results reported here should be taken as a *lower bound* for the information that can be recovered from a vectorized text representation.

Section 5.2.1 reports the results of the inversion when $\mathcal{A}$ is trained to infer words

(which appear in the plaintext) from the vectorized representation of sentences; Section 5.2.2 covers the case when $\mathcal{A}$ is extended to invert the encoding of longer pieces of text; Section 5.2.3 investigates whether or not it is possible to recover proper names form Sentence-BERT embeddings; finally, Section 5.2.4 analyzes how the inversion performance achieved by $\mathcal{A}$ scales with the number of training samples available.

5.2.1 Vectorized Sentences Inversion

The first analysis performed consists of inverting vectorized representations of single sentences obtained as described in Section 4.3.1. The inversion model $\mathcal{A}$ is trained, as described in Section 4.3.3, to invert the sentence representations produced by the following vectorization models: bag-of-words with tf-idf reweighting, hashing-based bag-of-words, and their reduced versions using LSA, Doc2Vec, and Sentence-BERT.

Even though the inversion model $\mathcal{A}$ stops the prediction whenever the `[EOS]` token is predicted (see Algorithm 1), the loss in Equation 4.4 does not directly penalize the model if it predicts more words than there are in the plaintext. To directly instruct the model to predict the `[EOS]` token and consequently stop the prediction to avoid increasing the number of false positives, the `[EOS]` token can be included in the target set of words $\mathcal{W} \subseteq \mathcal{V}_{\text{attack}}$. However, as analyzed in Appendix A.1, doing so yields a model $\mathcal{A}$ that has a lower number of false positives but a much higher number of false negatives, leading to high precision but low recall and consequently a low F_1 score. Without the inclusion of the `[EOS]` token, the model predicts it sometimes, but the final predictions usually contain several false positives. However, the resulting precision and recall are more balanced due to the lower number of false negatives, resulting in a higher F_1 score. Therefore, this section reports the results obtained when $\mathcal{A}$ was trained without including the `[EOS]` token in the set of words to predict.

To compare the performance between in-domain and cross-domain scenarios, the same test dataset $\mathcal{D}_{\text{target}}$, as described in Section 4.3.1, is used. Furthermore, when computing the performance metrics in the cross-domain scenario, only the words in $\mathcal{V}^{\text{in}}_{\text{attack}} \cap \mathcal{V}^{\text{cross}}_{\text{attack}}$ are considered.

Table 5.2 reports the aggregate inversion performance on the test dataset $\mathcal{D}_{\text{target}}$ in both the analyzed scenarios and using the metrics described in Section 4.3.2. For completeness, Appendix A.2 reports the cross-domain performance using all the words in $\mathcal{V}^{\text{cross}}_{\text{attack}}$.

The results of Table 5.2 confirm that the more complex the vectorization model $\Psi(\cdot)$ is, the less information can be recovered by $\mathcal{A}$ from the representation produced.

As expected, simple word counting, performed by bag-of-words or hashing-based

	Model	pre	rec	F_1	pre_w	rec_w	F_1^w
In-domain	Bag-of-words + tf-idf	94.37	87.05	89.05	94.27	84.73	87.34
	Hashing	94.72	87.27	89.33	94.15	84.86	87.21
	Reduced bag-of-words	62.00	61.80	59.40	57.99	55.20	53.48
	Reduced hashing	59.25	58.31	56.42	55.42	51.79	50.67
	Doc2Vec	49.51	55.10	49.56	54.23	55.65	52.55
	Sentence-BERT	41.53	45.47	41.36	41.55	42.04	39.58
Cross-domain	Bag-of-words + tf-idf	90.31	83.79	85.20	91.85	80.74	84.10
	Hashing	91.81	83.89	86.15	92.52	80.77	84.49
	Reduced bag-of-words	74.20	67.52	68.78	71.78	61.37	63.70
	Reduced hashing	71.13	65.12	65.97	68.56	58.91	60.79
	Doc2Vec	52.72	54.84	51.21	57.05	55.44	53.90
	Sentence-BERT	41.14	41.77	39.53	41.27	38.17	37.50

Table 5.2: Inversion results of vectorized sentences. Performance for in-domain and cross-domain scenarios is measured on the same test set. The performance metrics are precision, recall, F_1 score, and their generalized weighted variants. For each metric, the reported values correspond to the average value of all samples in the test set.

bag-of-words, leaks considerable information. With LSA on the bag-of-words representations, the amount of information recovered by $\mathcal{A}$ decreases while still being relatively high. However, it should be noted that, as reported in Section 5.1, when using LSA, the utility registered in the downstream task also decreases by a fair amount.

Notice that with feature hashing, the performance achieved in in-domain and cross-domain scenarios is slightly higher than that achieved with classical bag-of-words. However, when both representations are reduced using LSA, the situation is reversed, with $\mathcal{A}$ achieving higher performance on the reduced classical bag-of-words. This can be explained by the fact that when using feature hashing, the vectorization is not limited to a fixed vocabulary, as with classical bag-of-words. Therefore, if the plaintext has words that do not appear in the vocabulary of the vectorization model $\Psi(\cdot)$, those will not contribute to the final vector representation. Instead, with feature hashing, all words not excluded a priori (e.g., stop words) will contribute, possibly by colliding with other words, to the final vector representation, and assuming that unrelated words collide, $\mathcal{A}$ can guess the exact word using the context represented by the other words predicted. On the other hand, collisions of unrelated words caused by feature hashing make it difficult for LSA to map word co-occurrences to concepts. Therefore, information may get lost in the reduction, and consequently, $\mathcal{A}$ can recover a smaller amount of information.

Neural network based representations reduce the amount of information that $\mathcal{A}$

manages to recover. In particular, the representations produced by the deep model Sentence-BERT leak less information compared to the shallow model Doc2Vec. This can be explained by the fact that Doc2Vec encodes a text as a vector by optimizing the vector to be used to recover words that appear in the plaintext. In contrast, the multiple processing layers in the BERT architecture probably map plaintexts to high-level representations that encode the semantics of the plaintext rather than the words appearing in it. Nevertheless, the amount of information $\mathcal{A}$ recovers from Sentence-BERT embeddings is still quite impressive.

The performance achieved when $\mathcal{A}$ is trained using in-domain data instead of a large pool of cross-domain data is higher with bag-of-words, hashing-based bag-of-words, and Sentence-BERT. However, with LSA and Doc2Vec, the performance is higher using a large amount of cross-domain data, suggesting that the bottleneck is caused by the smaller amount of data available in the in-domain scenario.

On a further note, the inversion performance measured by the weighted variants of precision, recall, and F_1 score is slightly lower than that measured by the unweighted counterparts, the only outlier being Doc2Vec. This means that $\mathcal{A}$ manages to recover both words that appear with high frequency in $\mathcal{D}_{\text{target}}$ as well as those words that are rarer. However, it is possible that with more than 20k words in $\mathcal{V}_{\text{attack}}$, the weighted metrics would have told a different story.

Surprisingly, $\mathcal{A}$ recovers from Doc2Vec vectors rarer words more reliably than those that appear more frequently in $\mathcal{D}_{\text{target}}$, suggesting that the vector representation produced by Doc2Vec better encodes uncommon words rather than common ones.

To relate the performance metric values reported in Table 5.2 with the actual output of $\mathcal{A}$, Table 5.3 reports eight examples of Sentence-BERT embedding inversions which have been selected from *randomly* drawn samples from $\mathcal{D}_{\text{target}}$ and performed by $\mathcal{A}$ trained with in-domain data. In particular, Table 5.3 reports the plaintext x provided to the model $\Psi(\cdot)$ (Sentence-BERT in this case) and the output $\mathcal{W}_{\text{rec}}(x)$ of $\mathcal{A}$ when the vector $\Psi(x)$ is provided. Note that, as previously observed, this is the output of $\mathcal{A}$ in the most challenging setting when it achieves the lowest performance.

On a side note, notice that in the examples provided in Table 5.3, the number of predicted words ranges between 8 and 12 with several false positives (represented by words not in bold).

5.2.2 Vectorized Paragraphs Inversion

The experiments documented in Section 5.2.1 focus on the inversion of single sentences with a number of words (of at least two characters) between 8 and 40. There-

Plaintext	Inversion model prediction
l'ispezione si è conclusa con la richiesta alla compagnia irlandese di versare all'inps contributi per poco meno di dieci milioni di euro.	appena, **compagnia**, **conclusa**, **dieci**, **euro**, **inps**, **ispezione**, **milioni**, **poco**, utile, **versare**
certo, ci piacerebbe fare sempre di più ma per riuscirci c'è bisogno di favorevoli condizioni a contorno.	assai, **bisogno**, **certo**, **condizioni**, **fare**, farlo, possibile, pur, risultati, risultato, **sempre**
per l'autobrennero l'allungamento risulterebbe addirittura di 20 anni, con 3 miliardi di lavori.	**allungamento**, allungato, almeno, **anni**, anno, circa, concessione, **lavori**, **miliardi**, oltre, quasi, raddoppio
ciò che non cambia invece è che montepaschi potrebbe avere prospettive mediocri, perché opera in un'economia quasi immobile.	appare, azienda, cambierà, **ciò**, comunque, dimensioni, **invece**, **montepaschi**, **potrebbe**, può, risorse, situazione
ai fini del raggiungimento dei requisiti, nel rispetto dei limiti minimi di età e contribuzione, contano anche le frazioni d'anno.	**anno**, biennio, **contano**, **contribuzione**, **età**, incremento, limite, **limiti**, minimo, **raggiungimento**, **requisiti**
l'indice mib, maglia nera in europa, ha perso quasi il 5% (-4,3%) affossato dalle vendite sulle banche.	**banche**, borsa, **europa**, **indice**, italiane, **mib**, **perso**, piazza, **quasi**, **vendite**
esselunga, bernardo caprotti ritira la querela ai danni del figlio giuseppe.	**bernardo**, **caprotti**, confronti, denuncia, esposto, **esselunga**, **giuseppe**, risparmio
la commissione ue, però, ha dei dubbi sul percorso immaginato.	avere, **commissione**, **dubbi**, errore, **però**, piano, progetto, può, **ue**

Table 5.3: Inversion examples of Sentence-BERT sentence embeddings. The first column reports the text provided to the Sentence-BERT model. The second column reports the words (sorted alphabetically) inferred by $\mathcal{A}$ with the correctly predicted ones highlighted.

fore, it is interesting to analyze how the inversion performance changes when $\mathcal{A}$ is provided with the encoding of a longer piece of text instead of a single sentence.

To analyze how the inversion model $\mathcal{A}$ performs when provided with the encoding of a text comprising multiple sentences, the inversion models trained against the various vectorization models from Section 5.2.1, are applied to the vectorizations of the artificial test paragraphs with 128 to 168 words, constructed as described in Section 4.3.1.

In addition to using the trained models from Section 5.2.1, some inversion models

were trained *ad hoc* to invert paragraphs. In particular, in the in-domain scenario, $\mathcal{A}$ was trained to invert the representations of paragraphs produced by hashing-based bag-of-words and Sentence-BERT, which are, respectively, the worst and best performing models in terms of information leaked according to the results of Section 5.2.1. In the cross-domain scenario, only Sentence-BERT paragraph representations were used to train $\mathcal{A}$.

Table 5.4 reports the aggregate inversion performance, measured by precision, recall, and F_1 score (described in Section 4.3.2), on the paragraphs test dataset $\mathcal{D}_{\text{target}}$ in both the in-domain and cross-domain scenarios. In particular, Table 5.4 reports the inversion performance when $\mathcal{A}$ is provided with the encoding of the entire paragraph and when it is provided with the encoding of each sentence in the paragraph.

Notice that the test set used to obtain the inversion results of a paragraph by accessing the individual sentence encodings (sentences column) in Table 5.4 is the same as the one used to obtain the vectorized sentences inversion results reported in Table 5.2. The slightly different values of precision, recall, and F_1 score between the two tables are because each word appearing in a plaintext (either a sentence or a paragraph) gets only counted once, even if it appears multiple times in the plaintext.

Table 5.5 reports an example of an artificial paragraph encoded with Sentence-BERT and inverted by $\mathcal{A}$ trained to invert paragraphs using in-domain data.

From Table 5.4, we can observe that even though the inversion performance decreases for all models, a non-negligible amount of information can still be recovered.

In the in-domain scenario, when $\mathcal{A}$ is trained ad hoc on vectorized paragraphs, it achieves lower performance with respect to the cross-domain in the same setting (i.e., trained on vectorized paragraphs). This can be explained by the smaller amount of training data, as the training sentences are grouped to form training paragraphs.

Moreover, when $\mathcal{A}$ is trained with vectorized paragraphs, it performs better in inverting vectorized paragraphs, but it performs worse when inverting vectorized sentences. In particular, in the case of Sentence-BERT, much lower performance.

5.2.3 Name Recovery from Sentence-BERT Embeddings

In the previous sections, we established that information can be recovered from a vector of numbers representing a text and that the amount of such information depends on the vectorization model used. In particular, Sentence-BERT was shown to be the model — likely due to its deep architecture — least susceptible to such information leakage, even though a non-negligible amount of information can still be recovered from the representations it produces.

	Model	Paragraphs			Sentences		
		pre	rec	F_1	pre	rec	F_1
In-domain	Bag-of-words + tf-idf	39.50	32.62	35.56	95.09	81.44	87.38
	Hashing	38.71	32.19	34.98	95.35	81.56	87.55
	Hashing*	52.09	54.34	53.01	94.89	72.07	81.66
	Reduced bag-of-words	38.30	23.92	29.28	62.62	58.76	60.09
	Reduced hashing	38.59	22.19	28.00	59.67	55.07	56.77
	Doc2Vec	28.88	16.07	20.51	53.53	52.26	52.41
	Sentence-BERT	29.76	18.67	22.83	44.06	41.69	42.42
	Sentence-BERT*	28.57	20.96	24.01	27.62	24.31	25.60
Cross-domain	Bag-of-words + tf-idf	36.34	27.46	31.12	91.98	84.45	87.67
	Hashing	36.92	27.78	31.50	93.29	84.52	88.34
	Reduced bag-of-words	37.77	27.36	31.57	74.95	67.54	70.64
	Reduced hashing	36.43	23.97	28.78	71.48	64.67	67.48
	Doc2Vec	28.52	14.41	19.01	56.39	54.63	55.01
	Sentence-BERT	24.35	15.78	19.06	43.89	39.95	41.42
	Sentence-BERT*	29.02	20.94	24.16	30.56	29.29	29.63

* the inversion model was trained on vectorized paragraphs instead of sentences

Table 5.4: Inversion results of vectorized paragraphs. Performance for in-domain and cross-domain scenarios is measured on the same test set. The performance metrics are precision, recall, and F_1 score. For each metric, the reported values correspond to the average value of all samples in the test set. The paragraphs column reports the inversion performance when $\mathcal{A}$ is provided with the encoding of the entire paragraph. The sentences column reports the performance when $\mathcal{A}$ is provided with the encoding of each sentence in the paragraph.

A natural question is whether, given Sentence-BERT embeddings, it is possible to recover the names of people appearing in the corresponding plaintexts. If so, this would allow one to link the recovered information with the given person's name. In this section, we show that it is possible to recover names from Sentence-BERT embeddings, and even when the names are placed outside their usual context, the recovery is still viable.

Here, we focus on people's names because they are a simple example of sensitive information easily obtainable from the newspaper dataset. However, it goes without saying that the results obtained can be generalized to any other type of words occurring in $\mathcal{V}_{\text{attack}}$.

A list of sentences containing the name of any person in a given set was extracted from the test dataset $\mathcal{D}_{\text{target}}$ (71398 sentences). Since, as described in Section 4.3.1, $\mathcal{D}_{\text{target}}$ was generated from Economy news articles that naturally intersect with the Politics domain, the set of commonly occurring names selected for the analysis are Renzi (679 sentences), Draghi (529 sentences), Padoan (288 sentences), Berlusconi

Plaintext	Inversion model prediction
mondadori compra rcs libri «passaggio storico per l'editoria». nel consiglio della nuova rizzoli che sarà una spa, siederanno gian arturo ferrari, presidente, antonio porro, oddone pozzi, enrico selva coddè e paolo mieli, per quest'ultimo un riconoscimento del suo ruolo di presidente di rcs libri. come previsto dagli accordi siglati nell'ottobre scorso, il perimetro dell'acquisizione perfezionata ieri che comprende le partecipazioni e la titolarità di tutti i marchi. sulla base di specifiche clausole del contratto, il prezzo potrà essere soggetto ad aggiustamenti per un massimo di 5 milioni. c'è poi un «earn-out» in favore di rcs sino a 2,5 milioni, vincolo condizionato al conseguimento nel 2017 di determinati risultati. quanto alla squadra, selva coddè, amministratore delegato area trade di mondadori libri, guiderà la stessa divisione in rizzoli libri; antonio porro, amministratore delegato area educational, guiderà anche educational e internazionale illustrati di rizzoli.	accordo, **amministrazione**, andrea, anni, **azioni**, casa, circa, **consiglio**, **delegato**, dettagli, direttore, due, editore, editrice, esclusiva, **essere**, euro, fino, **libri**, mediagroup, **milioni**, **mondadori**, nuovi, offerta, **parte**, partecipazione, patto, **poi**, potrebbe, **potrà**, **presidente**, prevede, **previsto**, **rcs**, rispetto, **rizzoli**, sera, soci, solferino, stati, stesso, sviluppo, valorizzazione, via

Table 5.5: Inversion example of a Sentence-BERT paragraph embedding. The first column reports the artificial paragraph provided to the Sentence-BERT model. The second column reports the words (sorted alphabetically) inferred by $\mathcal{A}$ with the correctly predicted ones highlighted.

(287 sentences).

From all test sentences that contain the four selected names, new sentences are generated by substituting the original name with each of the other three names. For example, from the sentence "draghi: «il mandato delle banche centrali va visto in modo ampio»", the sentences "*name*: «il mandato delle banche centrali va visto in modo ampio»" for $name \in \{\text{renzi}, \text{padoan}, \text{berlusconi}\}$ are generated.

Substitution is performed on the last name and, when it appears, on the first name. Sentence-BERT is then used to obtain the embeddings of the sentences generated as described.

The inversion model $\mathcal{A}$ trained to invert Sentence-BERT embeddings using the cross-domain data $\mathcal{V}^{\text{cross}}_{\text{attack}}$ from Section 5.2.1 is used to infer the words appearing in the sentences[1].

Table 5.6 reports for each name along the columns, the fraction of times $\mathcal{A}$

[1]The cross-domain model is used because $\mathcal{V}^{\text{cross}}_{\text{attack}}$ also contains names from other domains. This fact is used in the following experiment.

correctly predicts the name[2] when it is replaced with one of the names along the rows. In particular, the values along the diagonal correspond to the accuracy of the prediction when the original sentences are used. In Table 5.6 it is also reported the number of sentences from $\mathcal{D}^{\text{cross}}_{\text{train}}$ (849692 sentences) each name occurs in.

It is interesting to observe that although Berlusconi is the most common name in $\mathcal{D}^{\text{cross}}_{\text{train}}$, it is the one for which $\mathcal{A}$ achieves the lowest accuracy. Furthermore, when Renzi replaces Draghi, $\mathcal{A}$ is more accurate than when the sentences with Draghi are left unchanged.

		From			
To	Samples	Berlusconi	Draghi	Padoan	Renzi
Berlusconi	17116	**77.27**	82.23	72.57	72.37
Draghi	287	68.33	81.29	81.94	77.64
Padoan	353	60.14	78.45	**88.54**	76.12
Renzi	15575	74.83	**82.42**	82.64	**83.01**

Table 5.6: Accuracies in the recovery of the last name from sentence-BERT embeddings with *in-domain* substitutions. For each name along the columns, it is reported the fraction of times $\mathcal{A}$ correctly recovers the last name when it is substituted with a name along the rows. Samples is the number of training examples ($\approx$ 850k) that contain that name.

The same experiment was repeated by substituting the previous four names with names from other domains such as Sport (e.g., Pellegrini), Arts (e.g., Pirandello), and Entertainment (e.g., Ligabue).

Similarly to before, Table 5.7 reports for each name along the columns, the fraction of times $\mathcal{A}$ correctly predicts the name when it is replaced with one of the names along the rows.

It is interesting to see how, regardless of the original sentence (i.e., the one containing one of the four names), the accuracy achieved by $\mathcal{A}$ when predicting the substituted name is very similar between the four original sentences.

These results suggest that name recovery from Sentence-BERT embedding is possible. The inversion does not only rely on the context represented by the other words appearing with the name, as we have seen that when the name is placed in a context, it usually does not belong to (e.g., Balotelli and European Central Bank), the name can still be recovered. Of course, the context may help $\mathcal{A}$ correctly recover words: for example, if the European Central Bank is mentioned, Draghi is a good guess of another possible word. This is the strength of $\mathcal{A}$ that uses its predictions

[2]Despite the substitution is performed on the full name, Table 5.6 reports the accuracies only for the last name.

		From			
To	Samples	Berlusconi	Draghi	Padoan	Renzi
Balotelli	1325	**39.50**	**45.75**	43.06	**48.78**
Benigni	295	29.18	32.89	30.90	35.71
Bocelli	92	5.69	7.18	7.99	11.63
Celentano	288	22.42	23.44	20.14	25.92
Fognini	232	17.08	20.79	18.40	26.53
Gallinari	129	4.27	4.54	5.21	8.57
Ligabue	122	13.52	14.37	19.44	20.61
Mancini	1212	21.71	30.81	25.69	30.82
Pellegrini	516	35.23	44.61	**48.96**	41.84
Pirandello	95	11.74	14.93	14.93	16.33

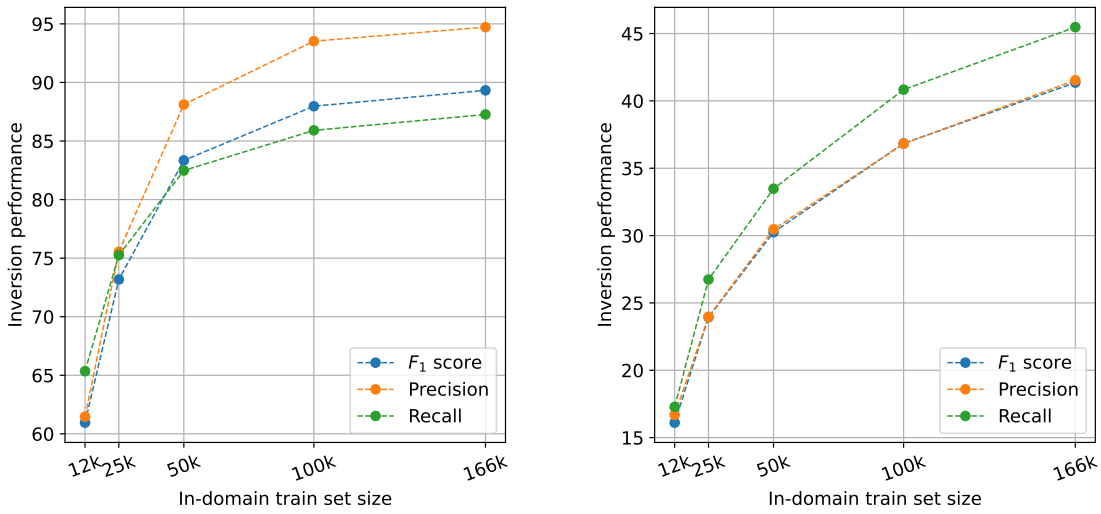
Table 5.7: Accuracies in the recovery of the last name from sentence-BERT embeddings with *cross-domain* substitutions. For each name along the columns, it is reported the fraction of times $\mathcal{A}$ correctly recovers the last name when it is substituted with a name along the rows. Samples is the number of training examples ($\approx$ 850k) that contain that name.

to bias its subsequent guesses.

5.2.4 Performance Scaling with the Amount of Training Data

It is interesting to analyze how the performance scales with the number of in-domain data available to the attacker. For this reason, the inversion model $\mathcal{A}$ was trained to invert vectorized sentences with a varying number of training examples. The same hyperparameter configuration, found as described in 4.3.3, was used. The model was probably more prone to overfit, given the reduced training data. It is very likely that, with a hyperparameter search for each training size, more performance could have been squeezed out of the inversion model. However, here we want to observe just the performance trend. In addition, to counterbalance the reduction in available training data, the number of epochs doubled from 100 to 200.

Figure 5.1 reports the inversion performance scaling for the hashing-based bag-of-words and Sentence-BERT embeddings, which were observed to be on opposite sides of the performance spectrum in Section 3.4.1.



(a) Hashing-based bag-of-words.

(b) Sentence-BERT.

Figure 5.1: Inversion performance scaling with the amount of training data. The inversion model is trained with a varying number of vectorized in-domain sentences.

Chapter 6

Conclusion

In this work, we examined an array of strategies for encoding text into vectors of features. Furthermore, we analyzed the amount of information an attacker can retrieve from these vectors by implementing a black-box inversion attack that only accesses the numerical vectors. The inversion attack was designed to be as universal as possible to be easily adapted to the various encoding strategies considered. Despite the attack model using a limited amount of information, the empirical results showed that a considerable amount of data can be reliably recovered even from the more advanced deep learning text representation models.

One of the objectives was to determine whether a partitioning approach consisting of locally computing the vectorizations of customers' text and training the downstream model on the cloud could protect the customer's privacy. Although this strategy appeared effective on the surface, the empirical results highlight the need for techniques specifically developed to address privacy leakage from vectorized text representations. For example, a possible approach worth exploring is anonymizing the vectors encoding texts via local differential privacy, as proposed in [41]. The inversion attack analyzed could then be used to investigate how possible mitigations affect privacy leakage other than the performance on the downstream task. For this purpose, Section 6.1 describes some of the improvements that could be applied to the attack model implemented in this work.

6.1 Inversion Attack Potential Improvements

In light of the results obtained and some observations mentioned in Chapter 4, here we list some potential improvements not analyzed in this work that could potentially improve the inversion attack analyzed.

- Use pre-trained word embeddings in place of randomly initialized ones in the

word embedding matrix W_2 (see Figure 4.2) to speed up the training of the attack model $\mathcal{A}$.

Moreover, suppose that we loosen the black-box access requirement. In that case, it is also possible for some models (e.g., Sentence-BERT or Doc2Vec) to use as attack model vocabulary $\mathcal{V}_{\text{attack}}$ the vectorization model vocabulary $\mathcal{V}$. Consequently, it is possible to initialize the word embedding matrix W_2 with the word embeddings used by the vectorization model.

- Change the loss in Equation 4.4 to better reward the attack model $\mathcal{A}$ whenever it correctly guesses words that occur with low frequency in the training dataset. For example, by weighting the estimated probabilities by $\mathcal{A}$ in a manner similar to that performed in Section 4.3.2 to appropriately weight the words in the generalized precision and recall metrics.

  It is possible that with this kind of modification, adding the `[EOS]` token to the target set of words would make the model predict fewer false positives. However, in Appendix A.1 it is shown that just adding the `[EOS]` token to the set of words to predict reduces the false positive predictions but also (with loss in Equation 4.4) increases the number of false negatives, leading to poor performance overall.

- As the cross-domain inversion model accesses a larger pool of training data, it could be interesting to investigate how its inversion performance changes when the number of words in $\mathcal{V}_{\text{attack}}$ increases beyond the 20k words considered here.

Appendix A

Additional Inversion Results

A.1 End-of-Sequence Token Included in the Set of Words to Predict

In Section 4.2.3, it was shown that the inversion model $\mathcal{A}$ stops the prediction whenever the `[EOS]` token is predicted (see Algorithm 1). The set prediction loss of Equation 4.4 does not penalize the model if it predicts more words than there are in the plaintext. Therefore, $\mathcal{A}$ rarely stops the prediction by predicting `[EOS]` before reaching the maximum number of prediction steps defined a priori. This consequently leads to many false positives (i.e., predicted words that do not appear in the plaintext).

It is possible to include the `[EOS]` token in the target set of words to predict $\mathcal{W} \subseteq \mathcal{V}_{\text{attack}}$. Doing so guides $\mathcal{A}$ to predict the `[EOS]` token and consequently stops the prediction to avoid increasing the number of false positives.

The inversion model $\mathcal{A}$ was trained with this modification using in-domain data to invert vectorized sentences produced by Sentence-BERT. The resulting precision, recall, and F_1 score averaged across all samples in the test set $\mathcal{D}_{\text{target}}$ are 64.05%, 18.75%, 27.20% respectively. For comparison, without the inclusion of `[EOS]` token, the measured precision, recall and F_1 score were 41.53%, 45.47%, 41.36% respectively (see Table 5.2).

Table A.1 reports the same eight examples as Table 5.3 of Sentence-BERT embedding inversions from $\mathcal{D}_{\text{target}}$ using $\mathcal{A}$ trained as just described. In particular, Table A.1 reports the plaintext x provided to the model $\Psi(\cdot)$ and the output $\mathcal{W}_{\text{rec}}(x) \setminus \{\texttt{[EOS]}\}$ of $\mathcal{A}$ when the vector $\Psi(x)$ is provided.

As can be observed from the performance metrics and the inversion examples, this strategy yields a model $\mathcal{A}$ that has a lower number of false positives but a much

higher number of false negatives, leading to increased precision but low recall, and consequently a low F_1 score.

Plaintext	Inversion model prediction
l'ispezione si è conclusa con la richiesta alla compagnia irlandese di versare all'inps contributi per poco meno di dieci milioni di euro.	**compagnia**, **euro**, **ispezione**, **milioni**, **poco**, **versare**
certo, ci piacerebbe fare sempre di più ma per riuscirci c'è bisogno di favorevoli condizioni a contorno.	**fare**
per l'autobrennero l'allungamento risulterebbe addirittura di 20 anni, con 3 miliardi di lavori.	**allungamento**, **anni**, quasi
ciò che non cambia invece è che montepaschi potrebbe avere prospettive mediocri, perché opera in un'economia quasi immobile.	
ai fini del raggiungimento dei requisiti, nel rispetto dei limiti minimi di età e contribuzione, contano anche le frazioni d'anno.	limite, **requisiti**
l'indice mib, maglia nera in europa, ha perso quasi il 5% (-4,3%) affossato dalle vendite sulle banche.	**banche**, **europa**, **indice**, **perso**, **quasi**
esselunga, bernardo caprotti ritira la querela ai danni del figlio giuseppe.	**bernardo**, **caprotti**, **esselunga**
la commissione ue, però, ha dei dubbi sul percorso immaginato.	**commissione**, **dubbi**, **però**, progetto, **ue**

Table A.1: Inversion examples of Sentence-BERT with EOS token included. The first column reports the text provided to the Sentence-BERT model. The second column reports the words inferred by $\mathcal{A}$ (excluding the EOS token) when the Sentence-BERT vector is provided. The predicted words are sorted alphabetically and the correctly predicted ones are highlighted.

A.2 Cross-domain Vectorized Sentences Inversion

In Section 5.2.1 the test dataset $\mathcal{D}_{\text{target}}$ generated as described in Section 4.3.1 was used to compare the inversion performance in the in-domain and cross-domain scenarios. That is, each vectorized text $\Psi(x)$ in the test dataset was assigned the set

of words to predict $\mathcal{W}(x) \subseteq \mathcal{V}^{\text{in}}_{\text{attack}}$. Therefore, if a word from $\mathcal{V}^{\text{cross}}_{\text{attack}} \setminus \mathcal{V}^{\text{in}}_{\text{attack}}$ appeared in x, it was not taken into account in the performance measures reported in the cross-domain section of Table 5.2. For completeness, here we report the performance in the cross-domain scenario using $\mathcal{D}'_{\text{target}}$ where the set of words to predict $\mathcal{W}(x) \subseteq \mathcal{V}^{\text{cross}}_{\text{attack}}$ is assigned to each vectorized text $\Psi(x)$ in the test dataset. In detail, $\mathcal{D}'_{\text{target}}$ is defined as

$$\mathcal{D}'_{\text{target}} = \left\{ (\Psi(x_i), \mathcal{W}(x_i)) \;\middle|\; \begin{array}{l} x_i \text{ test plaintext} \\ \mathcal{W}(x_i) \subseteq \mathcal{V}^{\text{cross}}_{\text{attack}} \end{array} \right\}$$

Table A.2 reports the aggregate inversion performance on the test dataset $\mathcal{D}'_{\text{target}}$ in the cross-domain scenario using the metrics described in Section 4.3.2.

Model	pre	rec	F_1	pre_w	rec_w	F_1^w
Bag-of-words + tf-idf	91.05	89.36	88.31	89.90	88.05	86.41
Hashing	92.56	89.44	89.34	91.76	88.04	87.77
Reduced bag-of-words	74.38	71.45	70.91	70.70	66.22	65.66
Reduced hashing	71.34	68.91	68.02	67.65	63.56	62.72
Doc2Vec	53.25	58.21	52.97	57.38	60.01	56.26
Sentence-BERT	41.34	44.04	40.65	41.35	40.96	38.90

Table A.2: Inversion results of vectorized sentences in the cross-domain scenario. The performance metrics are precision, recall, F_1 score, and their generalized weighted variants. For each metric, the reported values correspond to the average value of all samples in the test set.

Bibliography

[1] C. Arthur, *Tech giants may be huge, but nothing matches big data*, 2013. [Online]. Available: `https://www.theguardian.com/technology/2013/aug/23/tech-giants-data`.

[2] *What is natural language processing?*, 2020. [Online]. Available: `https://www.ibm.com/cloud/learn/natural-language-processing`.

[3] A. F. Westin, «Privacy and freedom», *Washington and Lee Law Review*, vol. 25, no. 1, p. 166, 1968.

[4] G. Sartor and F. Lagioia, «The impact of the general data protection regulation (gdpr) on artificial intelligence», *Directorate-General for Parliamentary Research Services of the Secretariat of the European Parliament. Brussels: European Union*, vol. 10, p. 293, 2020.

[5] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, «Membership inference attacks against machine learning models», in *2017 IEEE symposium on security and privacy (SP)*, 2017, pp. 3–18.

[6] V. Shejwalkar, H. A. Inan, A. Houmansadr, and R. Sim, «Membership inference attacks against nlp classification models», in *NeurIPS 2021 Workshop Privacy in Machine Learning*, 2021.

[7] N. Carlini, C. Liu, Ú. Erlingsson, J. Kos, and D. Song, «The secret sharer: Evaluating and testing unintended memorization in neural networks», in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 267–284.

[8] N. Carlini, F. Tramèr, E. Wallace, *et al.*, «Extracting training data from large language models», in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 2633–2650.

[9] C. Song and A. Raghunathan, «Information leakage in embedding models», in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 377–390.

[10] X. Pan, M. Zhang, S. Ji, and M. Yang, «Privacy risks of general-purpose language models», in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 1314–1331.

[11] *Text feature extraction*. [Online]. Available: `https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction`.

[12] C. Manning, P. Raghavan, and H. Schütze, «Matrix decompositions and latent semantic indexing», in *Introduction to Information Retrieval*. Cambridge University Press, 2008, ch. 18. [Online]. Available: `https://nlp.stanford.edu/IR-book/pdf/18lsi.pdf`.

[13] Q. Le and T. Mikolov, «Distributed representations of sentences and documents», in *International conference on machine learning*, PMLR, 2014, pp. 1188–1196.

[14] N. Reimers and I. Gurevych, «Sentence-bert: Sentence embeddings using siamese bert-networks», *arXiv preprint arXiv:1908.10084*, 2019.

[15] S. Palachy, *Document embedding techniques*, 2019. [Online]. Available: `https://towardsdatascience.com/document-embedding-techniques-fed3e7a6a25d`.

[16] *Truncated singular value decomposition and latent semantic analysis*. [Online]. Available: `https://scikit-learn.org/stable/modules/decomposition.html#lsa`.

[17] T. Mikolov, K. Chen, G. Corrado, and J. Dean, «Efficient estimation of word representations in vector space», *arXiv preprint arXiv:1301.3781*, 2013.

[18] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, «Distributed representations of words and phrases and their compositionality», *Advances in neural information processing systems*, vol. 26, 2013.

[19] A. M. Dai, C. Olah, and Q. V. Le, «Document embedding with paragraph vectors», *arXiv preprint arXiv:1507.07998*, 2015.

[20] J. H. Lau and T. Baldwin, «An empirical evaluation of doc2vec with practical insights into document embedding generation», *arXiv preprint arXiv:1607.05368*, 2016.

[21] R. Řehůřek and P. Sojka, «Software Framework for Topic Modelling with Large Corpora», in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, ELRA, 2010, pp. 45–50.

[22] T. Brown, B. Mann, N. Ryder, *et al.*, «Language models are few-shot learners», *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[23] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, «Bert: Pre-training of deep bidirectional transformers for language understanding», *arXiv preprint arXiv:1810.04805*, 2018.

[24] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, «Attention is all you need», *Advances in neural information processing systems*, vol. 30, 2017.

[25] A. Conneau and D. Kiela, «Senteval: An evaluation toolkit for universal sentence representations», *arXiv preprint arXiv:1803.05449*, 2018.

[26] Y. Wu, *Smart compose: Using neural networks to help write emails*, 2018. [Online]. Available: `https://ai.googleblog.com/2018/05/smart-compose-using-neural-networks-to.html`.

[27] M. Chen, J. Tworek, H. Jun, *et al.*, «Evaluating large language models trained on code», *arXiv preprint arXiv:2107.03374*, 2021.

[28] G. Barber, *Github's commercial ai tool was built from open source code*, 2021. [Online]. Available: `https://www.wired.com/story/github-commercial-ai-tool-built-open-source-code/`.

[29] M. Brubaker and S. Prince, *Tutorial #12: Differential privacy I: Introduction*, 2021. [Online]. Available: `https://www.borealisai.com/research-blogs/tutorial-12-differential-privacy-i-introduction/`.

[30] A. Narayanan and V. Shmatikov, «Robust de-anonymization of large sparse datasets», in *2008 IEEE Symposium on Security and Privacy (sp 2008)*, 2008, pp. 111–125.

[31] C. Dwork and A. Roth, *The Algorithmic Foundations of Differential Privacy*. Now Publishers, Inc., 2014, vol. 9, pp. 211–407. [Online]. Available: `https://www.cis.upenn.edu/~aaroth/Papers/privacybook.pdf`.

[32] D. Desfontaines, *A friendly, non-technical introduction to differential privacy*, Ted is writing things (personal blog), 2021. [Online]. Available: `https://desfontain.es/privacy/friendly-intro-to-differential-privacy.html`.

[33] S. De, L. Berrada, J. Hayes, S. L. Smith, and B. Balle, «Unlocking high-accuracy differentially private image classification through scale», *arXiv preprint arXiv:2204.13650*, 2022.

[34] B. McMahan and A. Thakurta, *Federated learning with formal differential privacy guarantees*, 2022. [Online]. Available: `https://ai.googleblog.com/2022/02/federated-learning-with-formal.html`.

[35] V. Ruehle, R. Sim, S. Yekhanin, *et al.*, *Privacy preserving machine learning: Maintaining confidentiality and preserving trust*, 2021. [Online]. Available: https://www.microsoft.com/en-us/research/blog/privacy-preserving-machine-learning-maintaining-confidentiality-and-preserving-trust.

[36] M. Abadi, A. Chu, I. Goodfellow, *et al.*, «Deep learning with differential privacy», in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 308–318.

[37] G. Sharma, N. Hegde, S. Prince, and M. Brubaker, *Tutorial #13: Differential privacy II: Machine learning and data generation*, 2021. [Online]. Available: https://www.borealisai.com/research-blogs/tutorial-13-differential-privacy-ii-machine-learning-and-data-generation/.

[38] D. Yu, S. Naik, A. Backurs, *et al.*, «Differentially private fine-tuning of language models», *arXiv preprint arXiv:2110.06500*, 2021.

[39] M. Senge, T. Igamberdiev, and I. Habernal, «One size does not fit all: Investigating strategies for differentially-private learning across nlp tasks», *arXiv preprint arXiv:2112.08159*, 2021.

[40] I. Habernal, «When differential privacy meets nlp: The devil is in the detail», *arXiv preprint arXiv:2109.03175*, 2021.

[41] O. Feyisetan and S. Kasiviswanathan, «Private release of text embedding vectors», in *Proceedings of the First Workshop on Trustworthy Natural Language Processing*, 2021, pp. 15–27.

[42] X. Yue, M. Du, T. Wang, Y. Li, H. Sun, and S. S. Chow, «Differential privacy for text analytics via natural text sanitization», *arXiv preprint arXiv:2106.01221*, 2021.

[43] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, «Scikit-learn: Machine learning in Python», *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[44] E. L. Bird Steven and E. Klein, *Natural Language Processing with Python*. O'Reilly Media Inc., 2009.

[45] *Italian wikipedia*, Dump of the 01/06/2022, Wikimedia Foundation, 2022. [Online]. Available: https://dumps.wikimedia.org/itwiki/.

[46] S. Schweter, *Italian bert and electra models*, 2020. [Online]. Available: https://github.com/dbmdz/berts#italian-bert.

[47] P. May, *Machine translated multilingual sts benchmark dataset*, 2021. [Online]. Available: `https://github.com/PhilipMay/stsb-multi-mt`.

[48] N. Rieke, J. Hancox, W. Li, *et al.*, «The future of digital health with federated learning», *NPJ digital medicine*, vol. 3, no. 1, pp. 1–7, 2020.

[49] *Machine learning glossary*. [Online]. Available: `https://developers.google.com/machine-learning/glossary`.

[50] A. Paszke, S. Gross, F. Massa, *et al.*, «Pytorch: An imperative style, high-performance deep learning library», in *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, pp. 8024–8035.

[51] A. V. Joshi, «Amazon's machine learning toolkit: Sagemaker», in *Machine learning and artificial intelligence*, Springer, 2020, pp. 233–243.

[52] Amazon, *Amazon SageMaker Python SDK*, Amazon. [Online]. Available: `https://github.com/aws/sagemaker-python-sdk`.